

(12) UK Patent Application (19) GB (11) 2 355 837 (13) A

(43) Date of A Publication 02.05.2001

(21) Application No 0025202.3

(22) Date of Filing 13.10.2000

(30) Priority Data

(31) 9925576

(32) 28.10.1999

(33) GB

(71) Applicant(s)

Canon Kabushiki Kaisha

(Incorporated in Japan)

30-2 3-chome, Shimomaruko, Ohta-ku, Tokyo, Japan

(72) Inventor(s)

Philip Neil Garner

Jason Peter Andrew Charlesworth

Asako Higuchi

(74) Agent and/or Address for Service

Beresford & Co

2-5 Warwick Court, High Holborn, LONDON,
WC1R 5DH, United Kingdom

(51) INT CL⁷

G10L 15/12 // G10L 15:12

(52) UK CL (Edition S)

G4R RHB RPD RRL R1F

G4A AKS

U1S S1032 S2105 S2106 S2124 S2137 S2141

(56) Documents Cited

GB 2302199 A

(58) Field of Search

UK CL (Edition R) G4R RHB RPP RRL RRM

INT CL⁷ G06K 9/00 9/36 9/46 9/54 9/60 9/62 9/64 9/68

9/78 9/80 , G10L 15/00 15/02 15/08 15/12 15/14

Online:WPI, EPODOC, JAPIO

(54) Abstract Title

Pattern matching

(57) A system is provided for comparing an input query with a number of stored annotations to identify information to be retrieved from a database 29. The comparison technique divides the input query into a number of fixed-size fragments and identifies how many times each of the fragments occurs within each annotation. The frequencies of occurrence of the fragments in both the query and the annotation are then compared using an equation derived from a multinomial model to provide a measure of the similarity between the query and the annotation. The information to be retrieved is then determined from the similarity measures obtained for all the annotations.

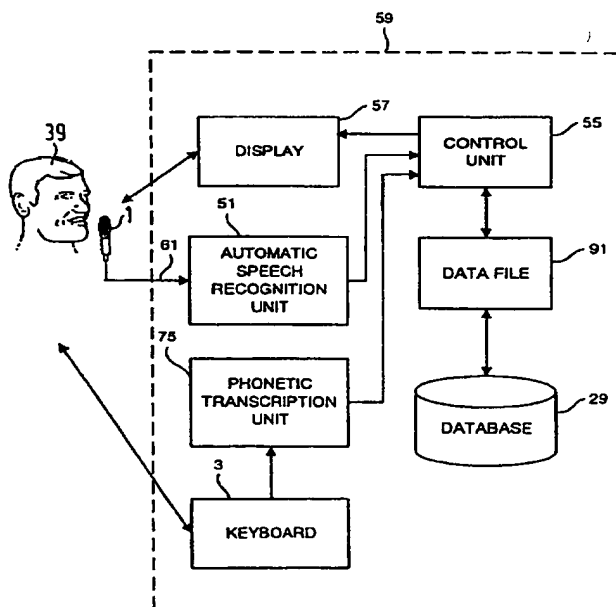


Fig. 1

GB 2 355 837 A

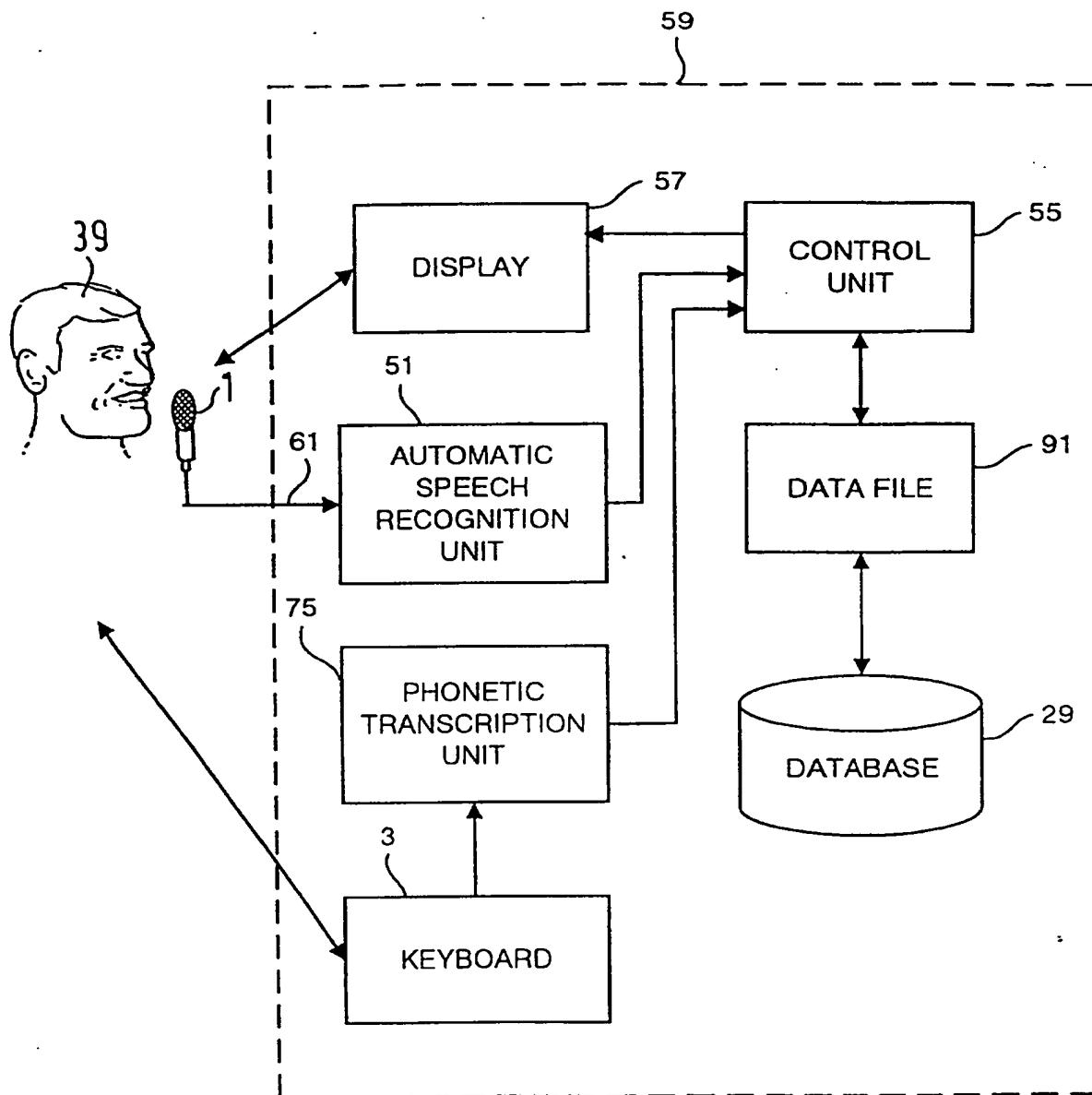


Fig. 1

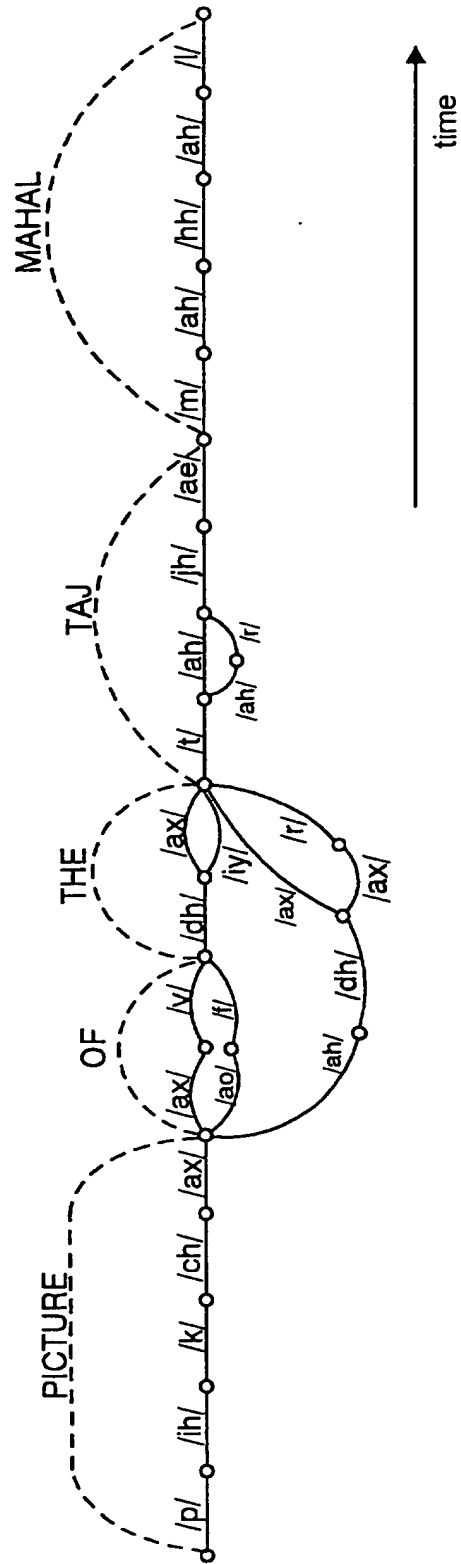


Fig. 2

3/22

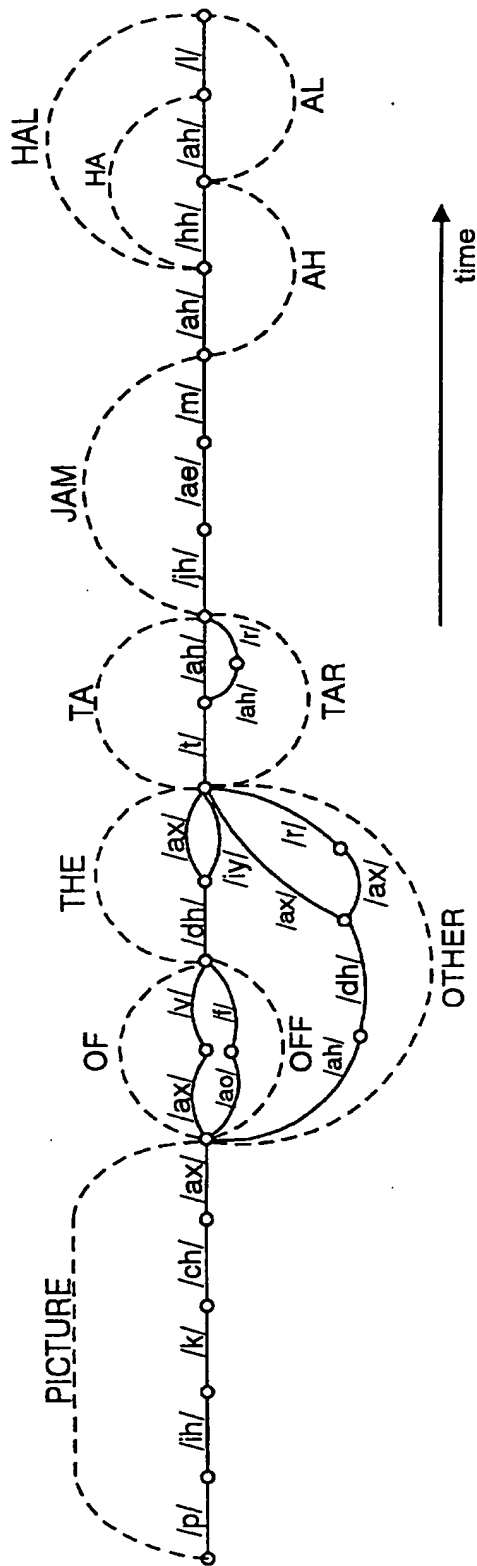


Fig. 3

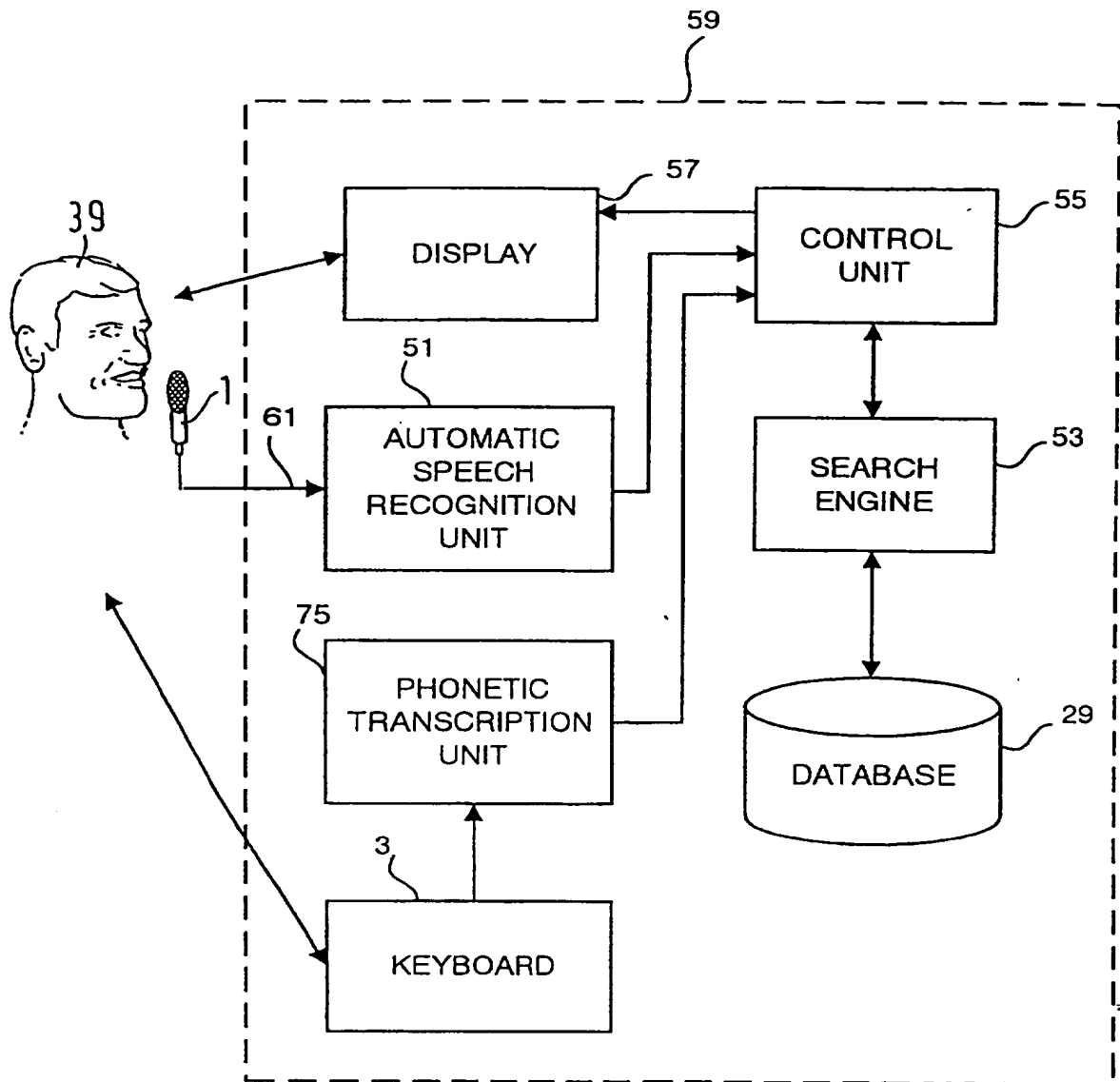


Fig. 4

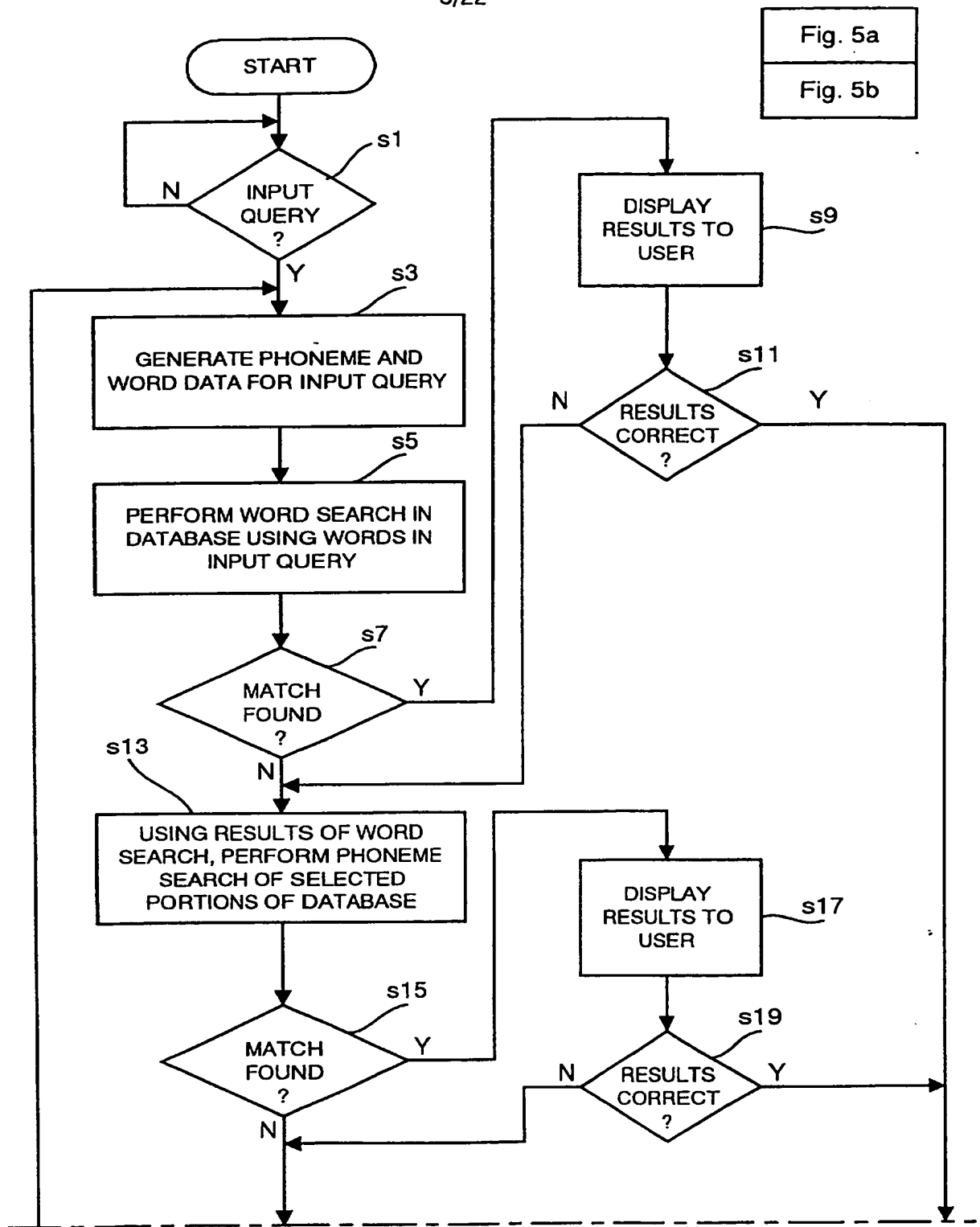


Fig. 5a

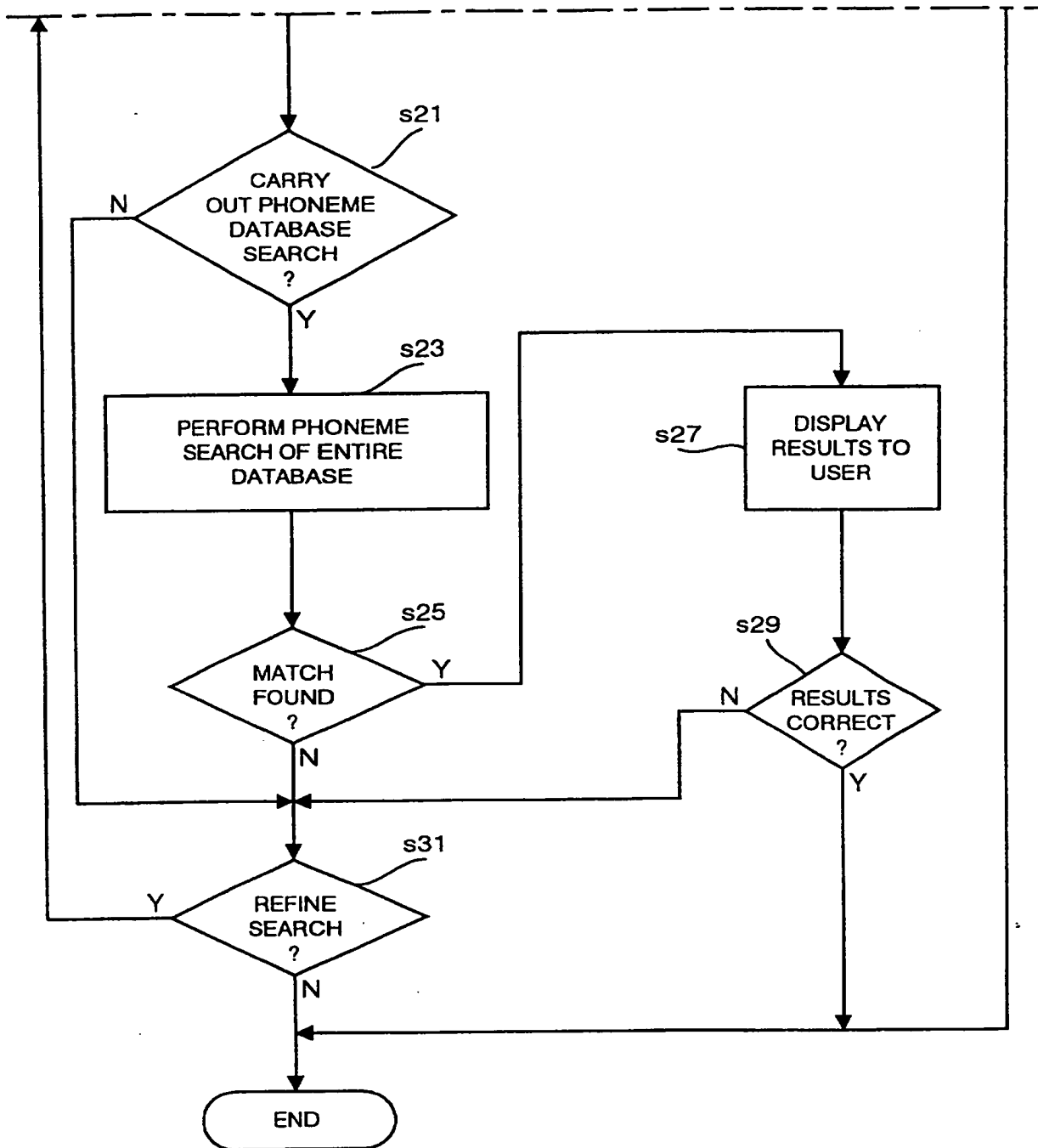


Fig. 5b

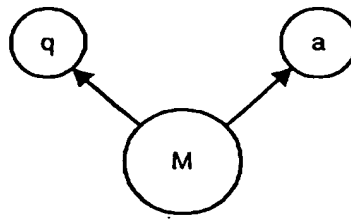


Fig. 6a

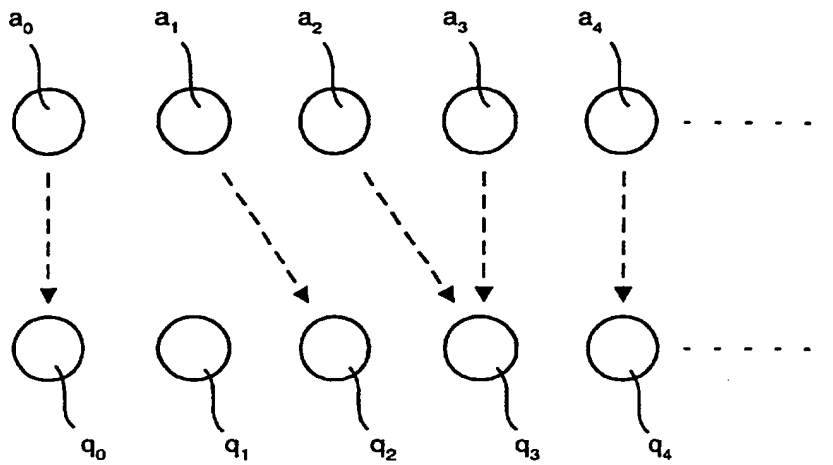


Fig. 6b

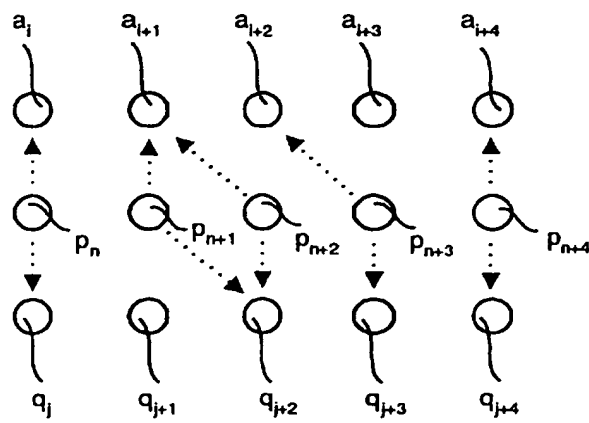
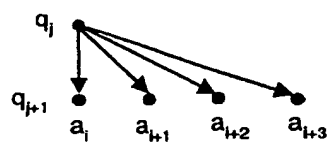
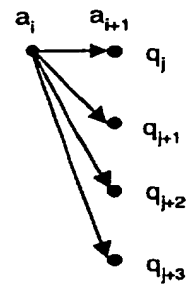
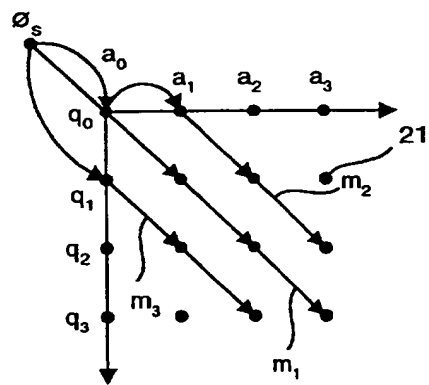
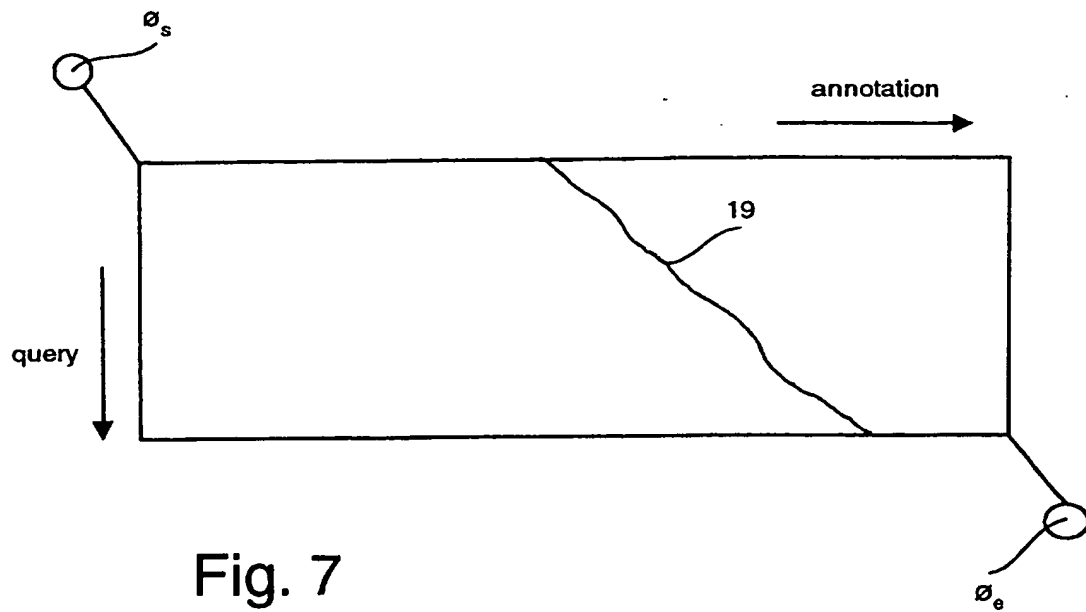


Fig. 6c



9/22

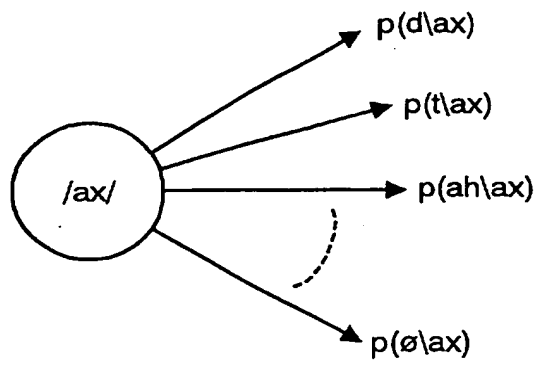


Fig. 10

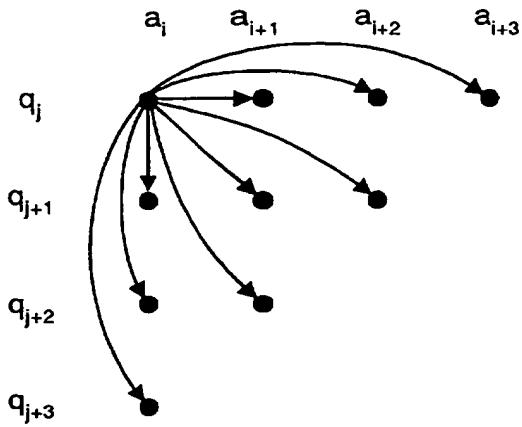


Fig. 11

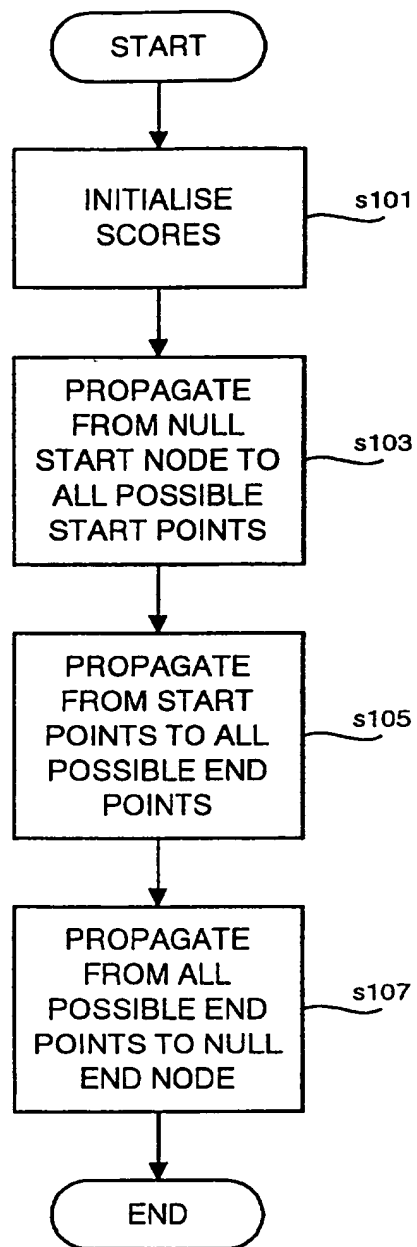


Fig. 12

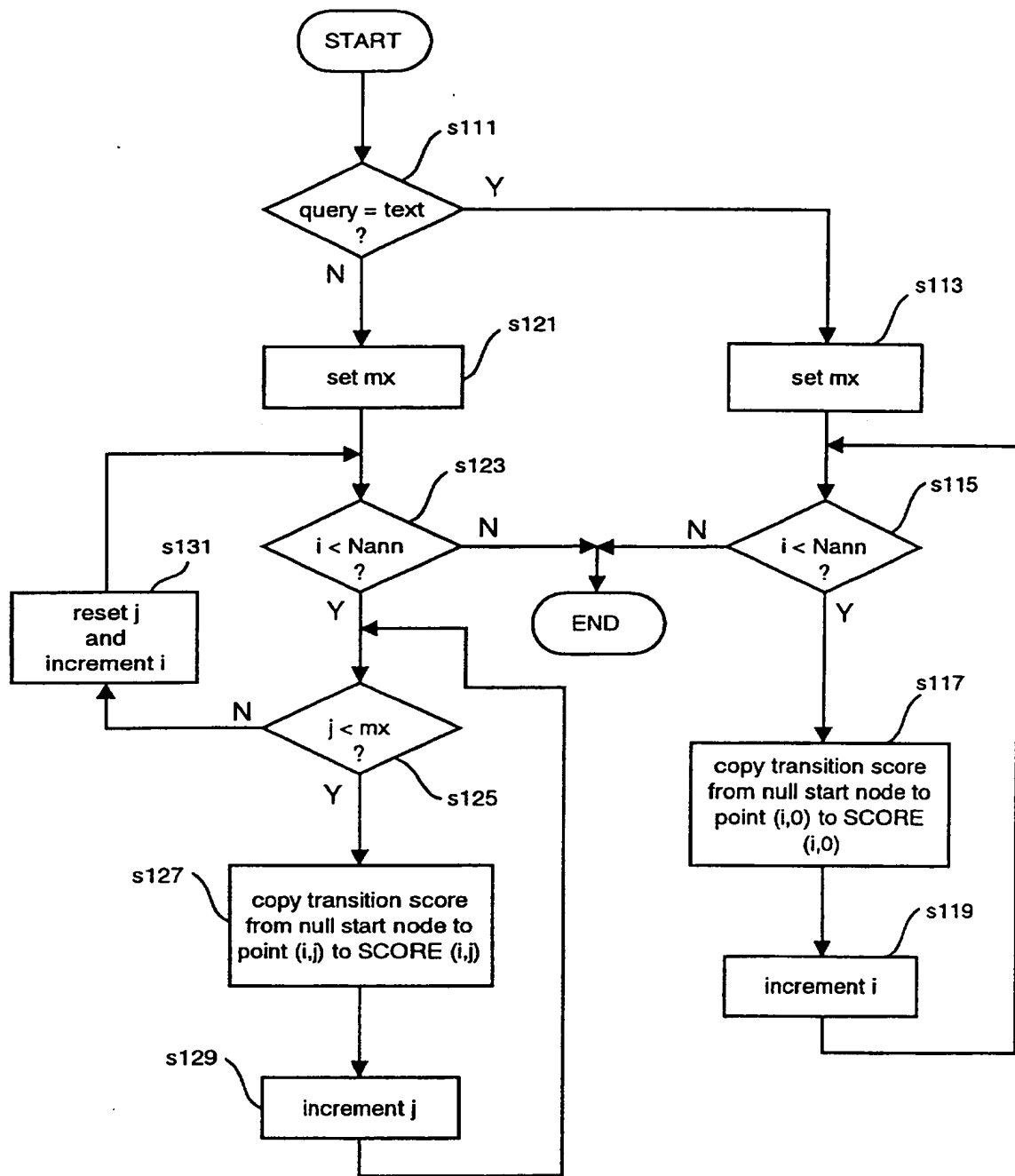


Fig. 13

12/22

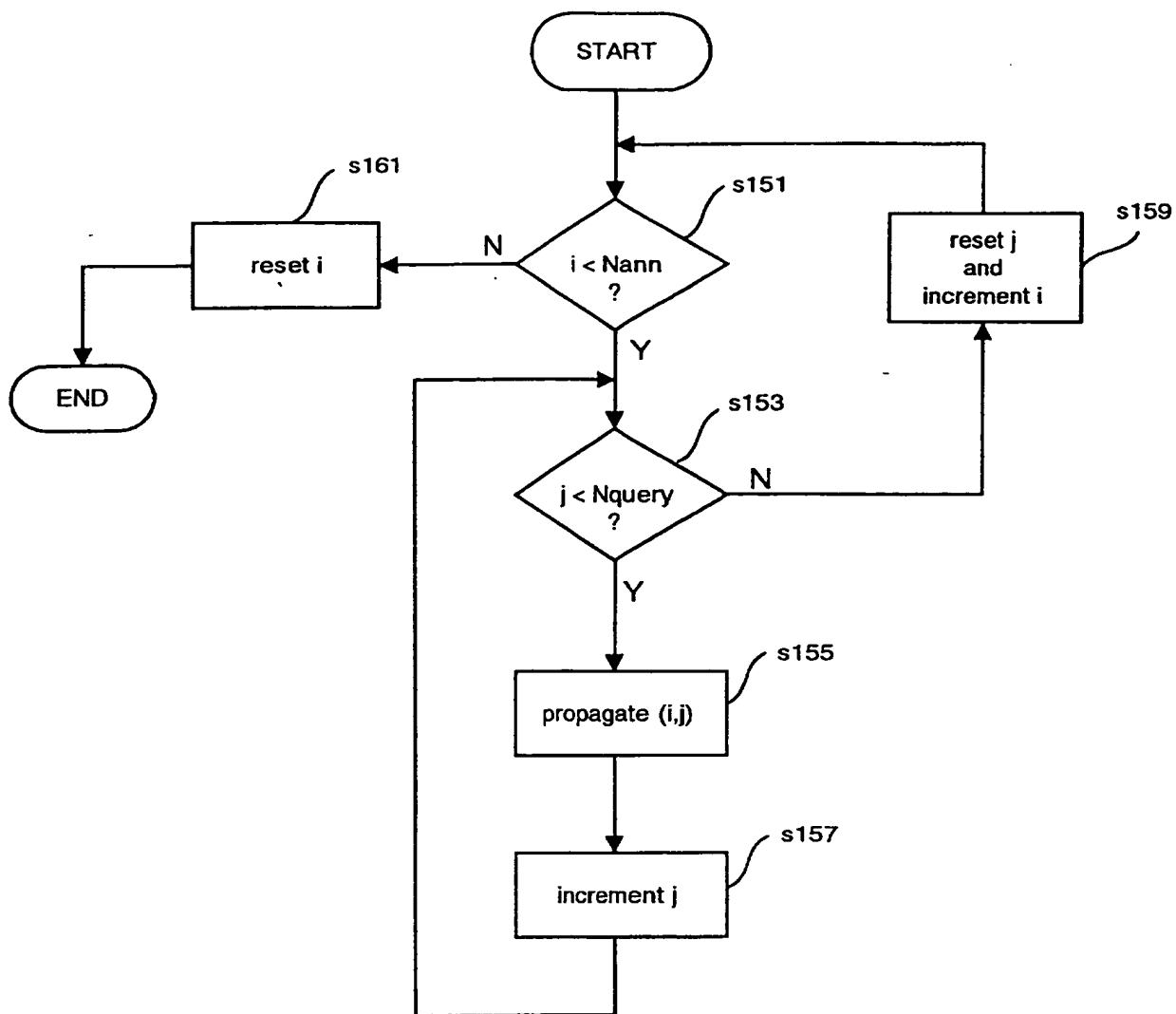


Fig. 14

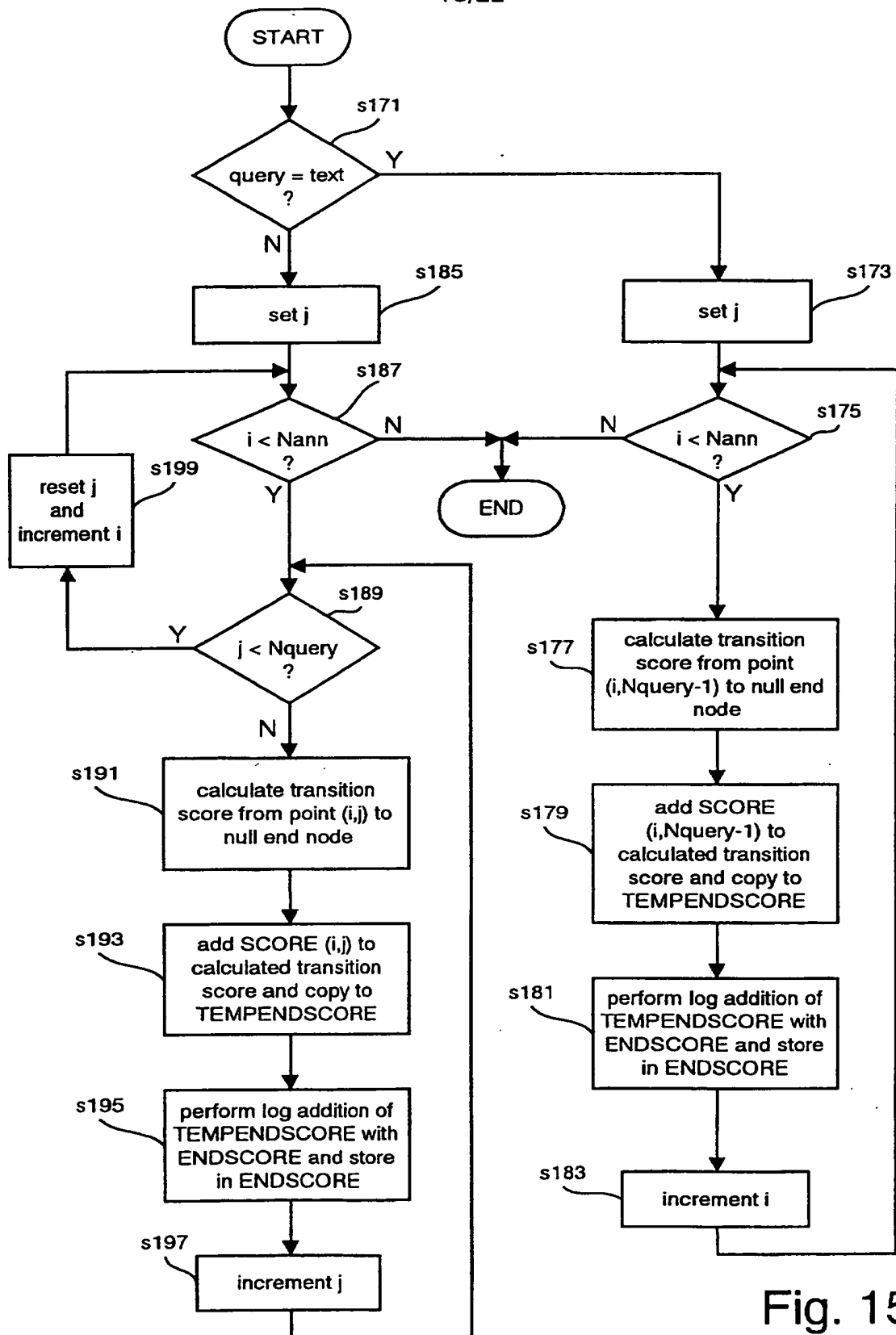


Fig. 15

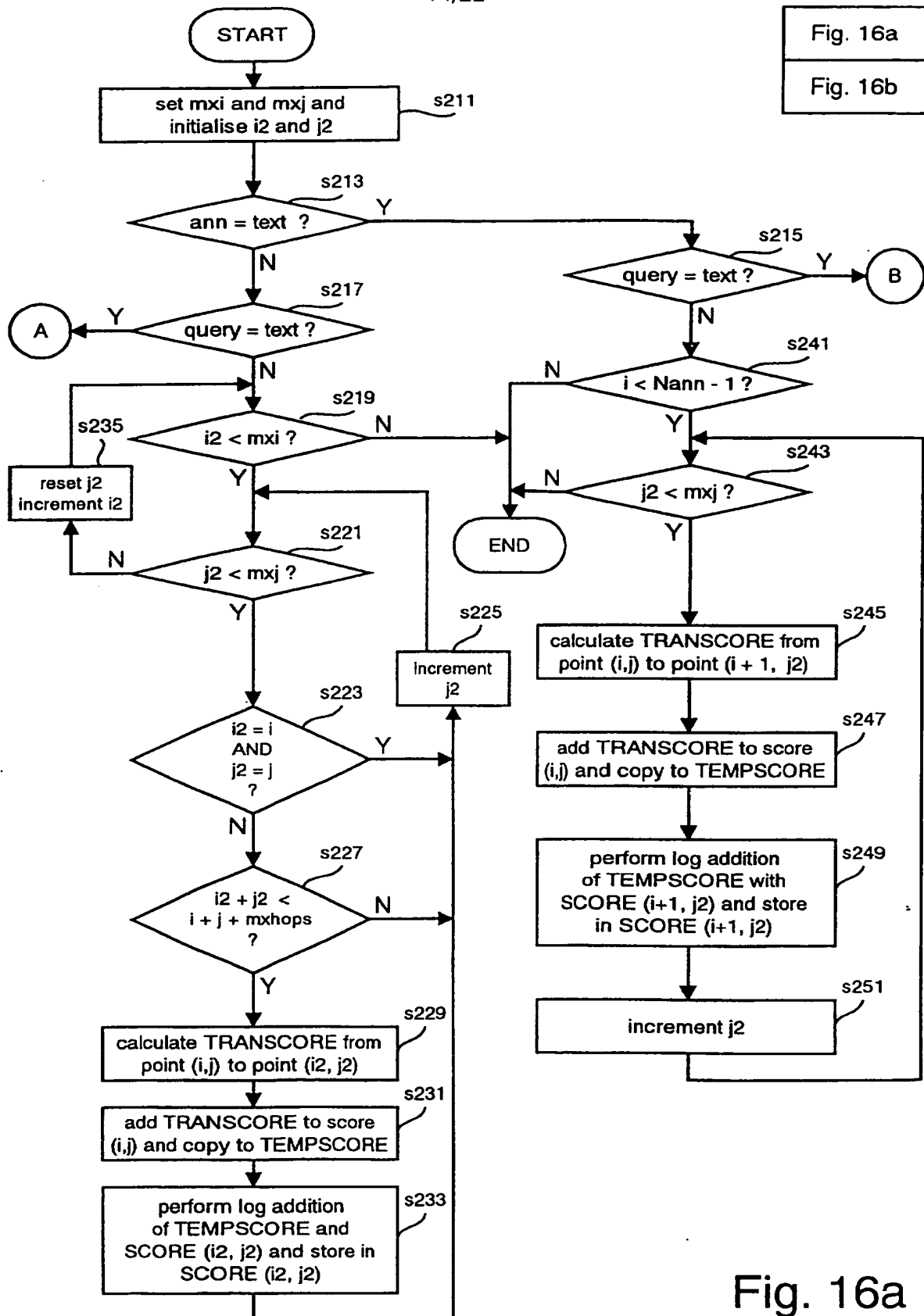


Fig. 16a

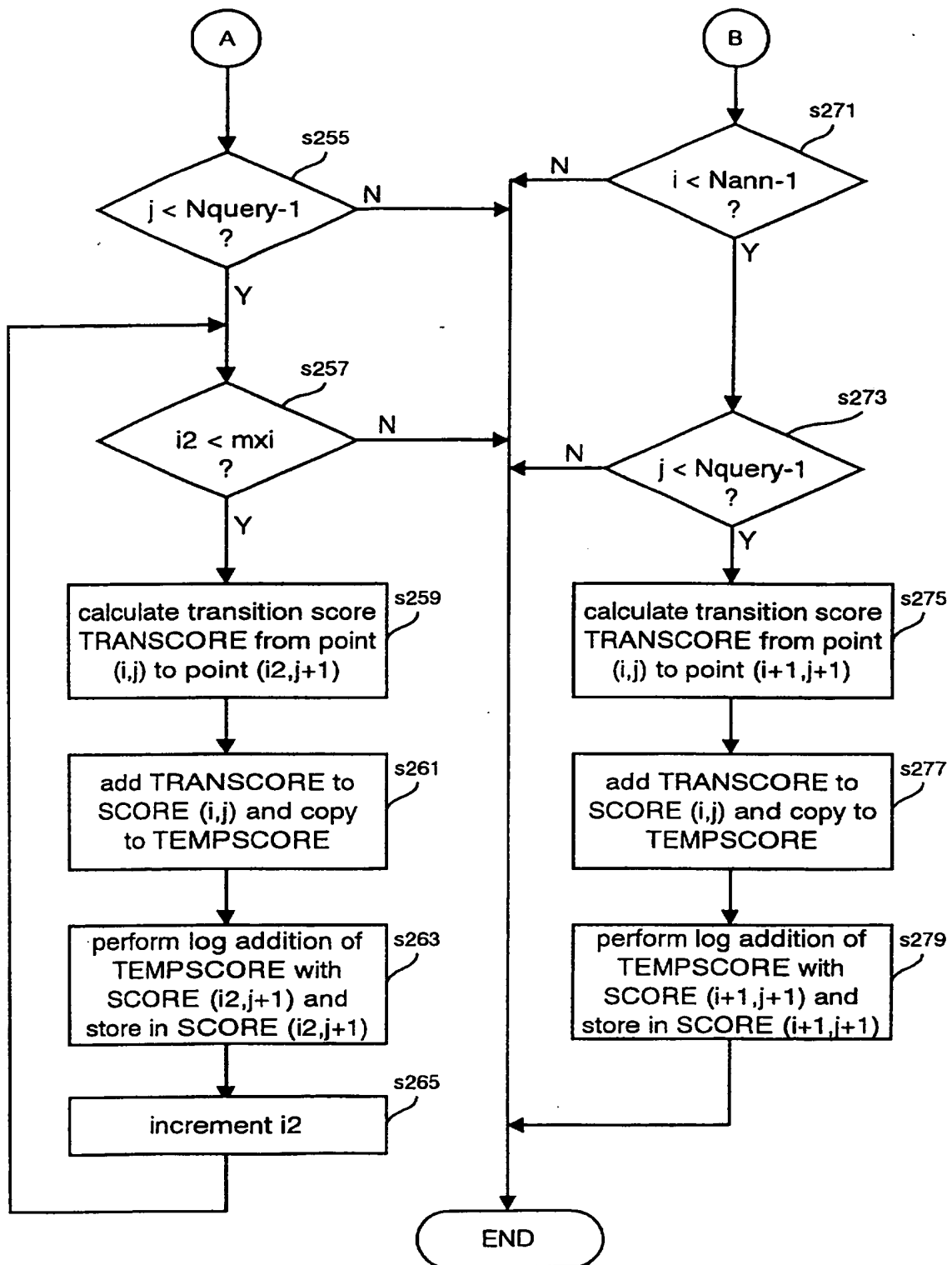


Fig. 16b

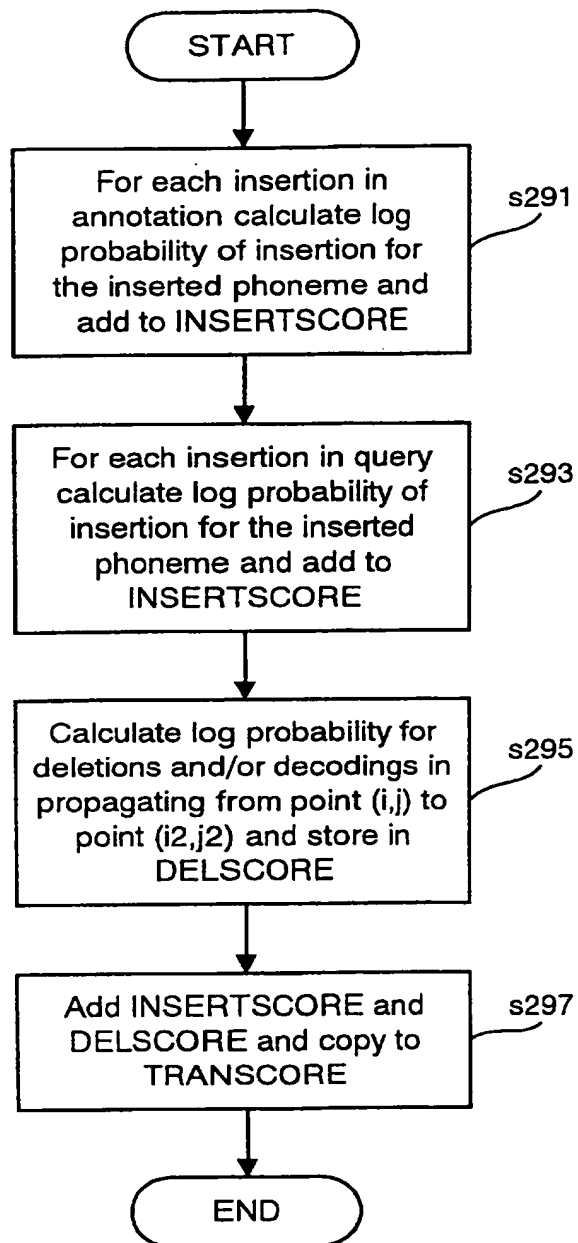


Fig. 17

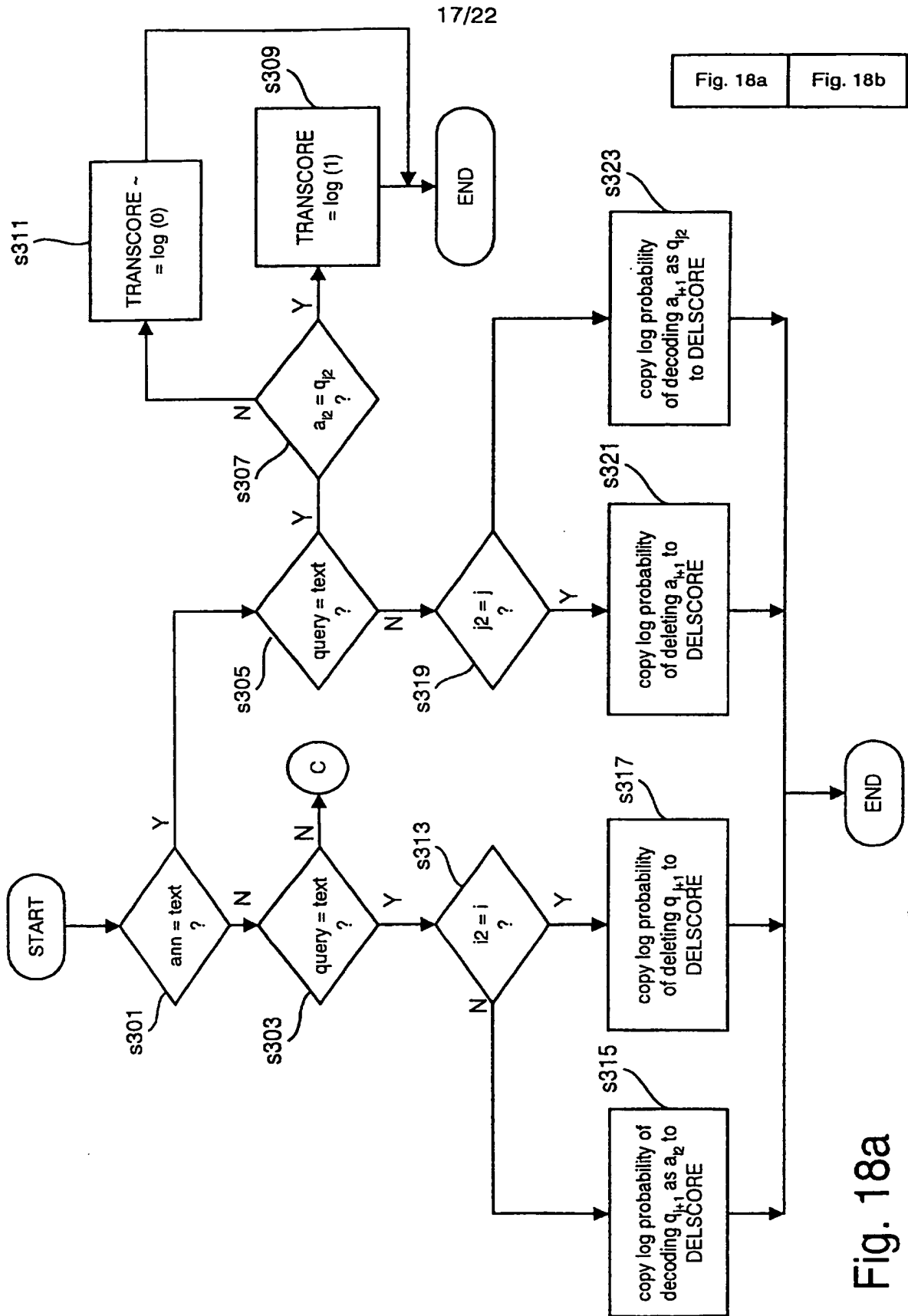
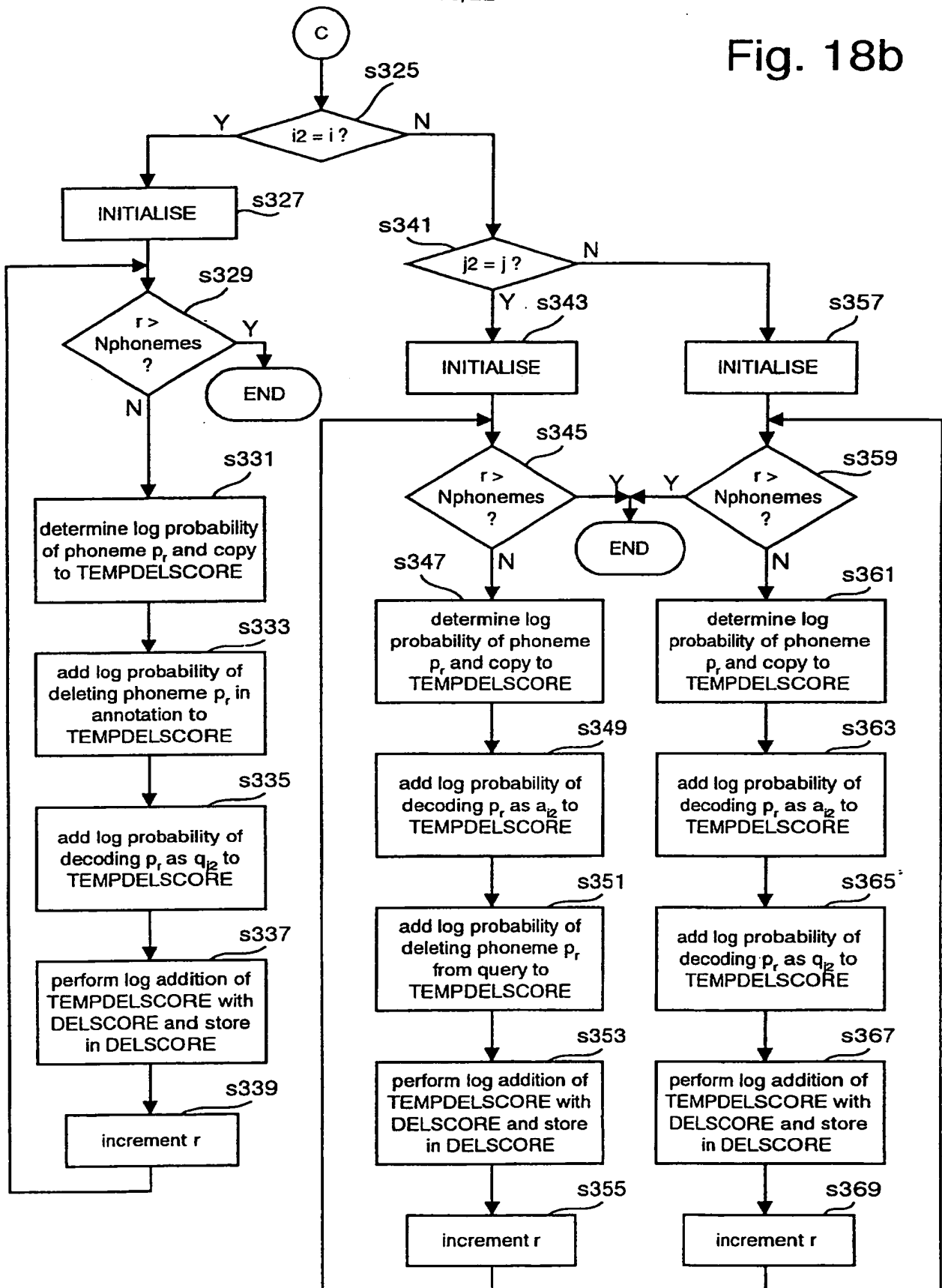


Fig. 18b



19/22

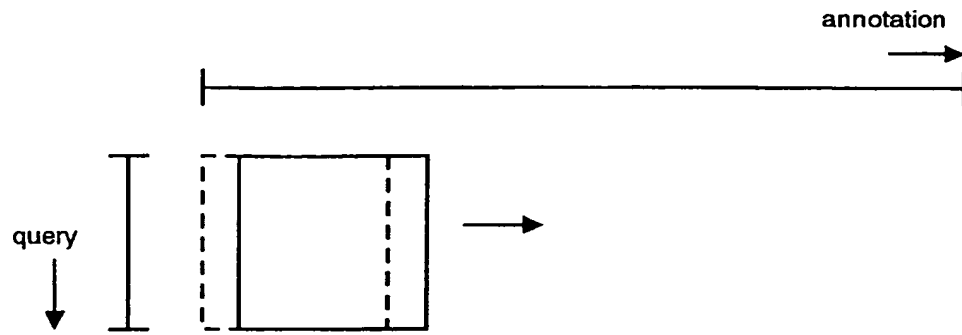


Fig. 19a

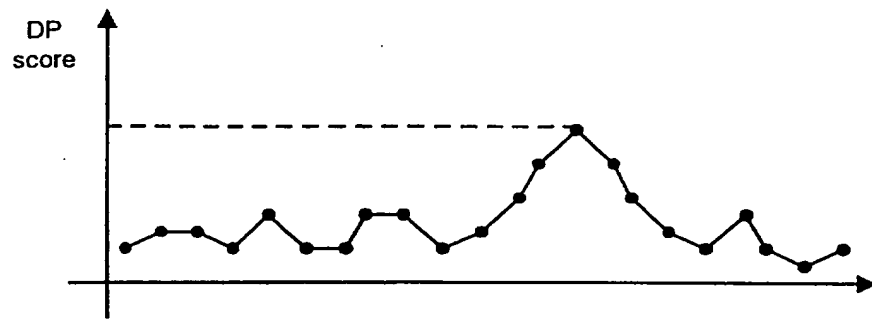


Fig. 19b

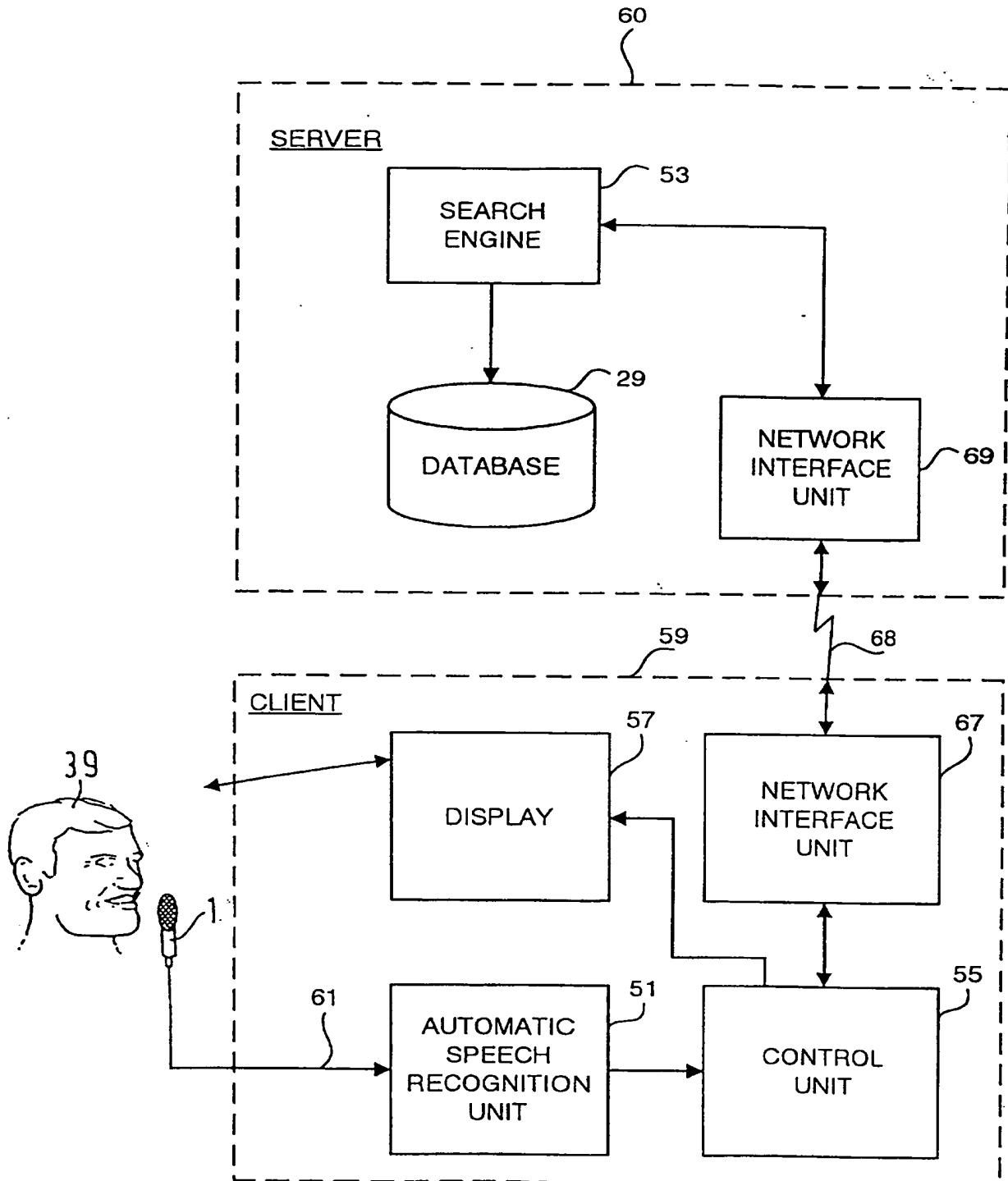


Fig. 20

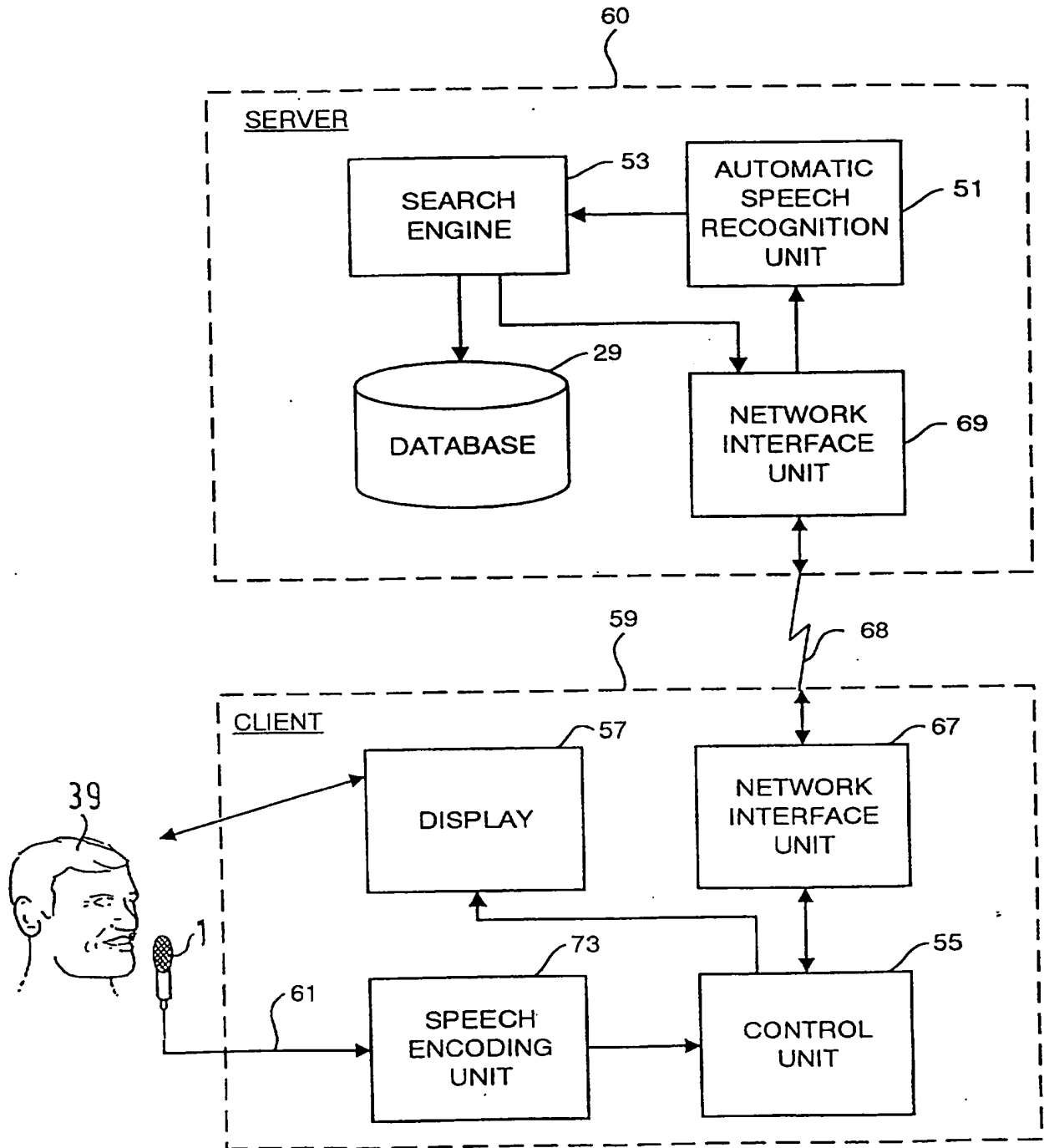


Fig. 21

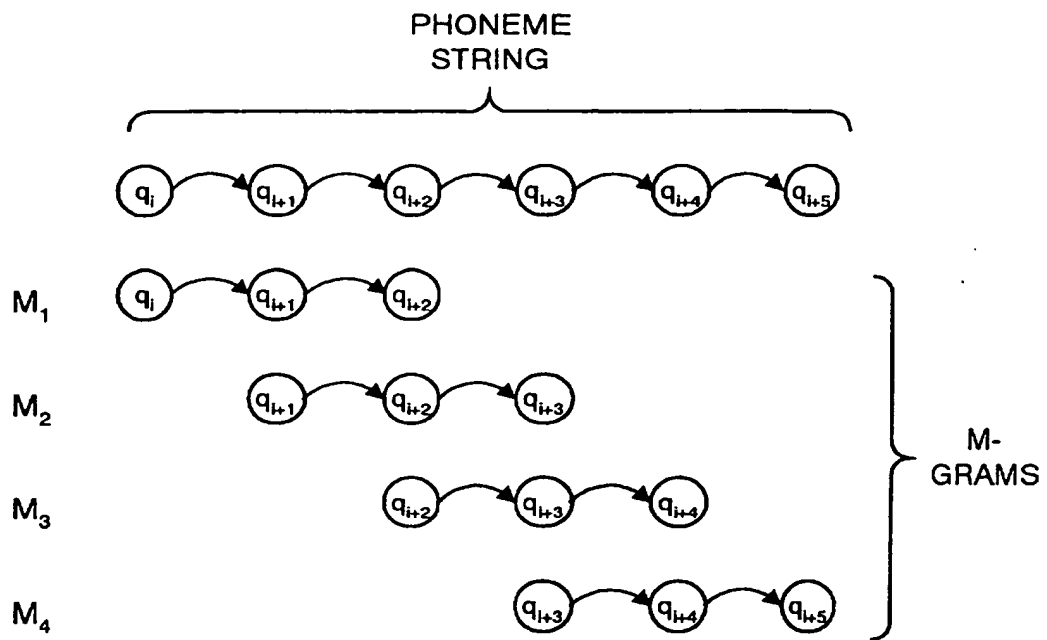


Fig. 22

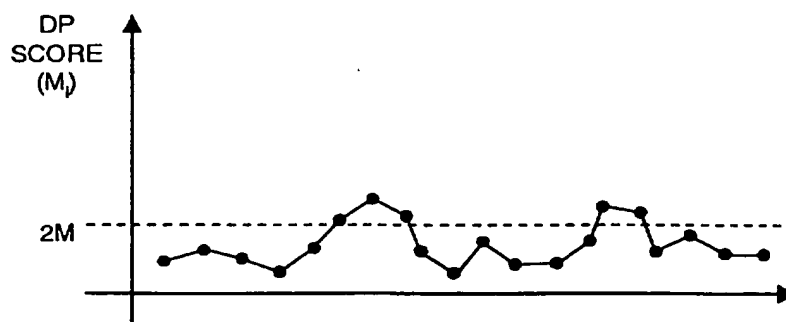


Fig. 23

PATTERN MATCHING METHOD AND APPARATUS

The present invention relates to an apparatus and method for matching sequences of phonemes or the like. The invention can be used to search a database of data files having associated phonetic annotations, in response to a user's input query. The input query may be a voiced or typed query.

Databases of information are well known and suffer from the problem of how to locate and retrieve the desired information from the database quickly and efficiently. Existing database search tools allow the user to search the database using typed keywords. Whilst this is quick and efficient, this type of searching is not suitable for various kinds of databases, such as video or audio databases.

A recent proposal has been made to annotate such video and audio databases with a phonetic transcription of the speech content in the audio and video files, with subsequent retrieval being achieved by comparing a phonetic transcription of a user's input query with the phoneme annotation data in the database. The technique proposed for matching the sequences of phonemes firstly defines a set of features in the query, each feature being taken as an overlapping fixed size fragment from the phoneme string, it then identifies the frequency of occurrence of the features in both the query and the annotation and then finally determines a measure of the similarity between the query and the annotation using a cosine measure of these frequencies of occurrences. One advantage of this kind of phoneme comparison technique is that it can cope with situations where the sequence of

words of the query do not exactly match the sequence of words of the annotation. However, it suffers from the problem that it is prone to error especially when the query and the annotation are spoken at different speeds and if there are any deletions of parts of words from the query, but not from the annotation, or vice versa.

The present invention aims to provide an alternative technique for matching sequences of phonemes.

According to one aspect, the present invention provides a feature comparison apparatus comprising means for receiving first and second sequences of features; means for identifying a plurality of different first sub-sequences of features within said first sequence of features; first obtaining means for obtaining the number of times each of said different first sub-sequences occurs within said first sequence of features; second obtaining means for obtaining the number of times each of said different first sub-sequences occurs within said second sequence of features; and means for computing a similarity score representative of the similarity between the first and second sequences of features using the numbers obtained from said first and second obtaining means; characterised in that the apparatus further comprises third obtaining means for obtaining the total number of sub-sequences of features in said second sequence; and in that said computing means comprises: first computing means for computing a measure of the probability of each of said first sub-sequences occurring in said second sequence of features using the numbers obtained from said second obtaining means and the number obtained from said third obtaining means; and second computing means for computing said similarity score by

taking products of said computed probability measures in dependence upon said numbers obtained from said first obtaining means.

5 According to another aspect, the present invention provides an apparatus for searching a database comprising a plurality of information entries to identify information to be retrieved therefrom, each of said plurality of information entries having an associated
10 annotation comprising a sequence of speech features, the apparatus comprising: means for receiving an input query comprising a sequence of speech features; the above feature comparison apparatus for comparing said query sequence of speech features with the speech features of
15 each annotation to provide a set of comparison results; and means for identifying said information to be retrieved from said database using said comparison results; characterised in that said feature comparison apparatus has a plurality of different comparison modes
20 of operation and in that the apparatus further comprises: means for determining (i) if the query sequence of speech features was generated from an audio signal or from text; and (ii) if the sequence of speech features of a current annotation was generated from an audio signal or from
25 text, and for outputting a determination result; and means for selecting, for the current annotation, the mode of operation of said feature comparison apparatus in dependence upon said determination result.

30 Exemplary embodiments of the present invention will now be described with reference to Figures 1 to 23, in which:

Figure 1 is a schematic block diagram illustrating a user terminal which allows the annotation of a data file with

annotation data generated from a typed or voiced input from a user;

5 Figure 2 is a schematic diagram of phoneme and word lattice annotation data which is generated from a typed input by the user for annotating the data file;

10 Figure 3 is a schematic diagram of phoneme and word lattice annotation data which is generated from a voiced input by the user for annotating the data file;

15 Figure 4 is a schematic block diagram of a user's terminal which allows the user to retrieve information from the database by a typed or voice query;

Figure 5a is a flow diagram illustrating part of the flow control of the user terminal shown in Figure 4;

20 Figure 5b is a flow diagram illustrating the remaining part of the flow control of the user terminal shown in Figure 4;

25 Figure 6a is a schematic diagram which shows an underlying statistical model which is assumed to have generated both the query and the annotation;

30 Figure 6b is a schematic diagram which shows a first sequence of phonemes representative of a typed input and a second sequence of phonemes representative of a user's voice input, and which illustrates the possibility of there being phoneme insertions and deletions from the user's voice input relative to the typed input;

Figure 6c is a schematic diagram which shows a first and

second sequence of phonemes, each representative of a voiced input and a third sequence of phonemes representative of a canonical sequence of phonemes corresponding to what was actually said in the
5 corresponding voiced inputs, and which illustrates the possibility of there being phoneme insertions and deletions from the two voiced inputs relative to the corresponding canonical sequence of phonemes;

10 Figure 7 schematically illustrates a search space created by the sequence of annotation phonemes and the sequence of query phonemes together with a start null node and an end null node;

15 Figure 8 is a two dimensional plot with the horizontal axis being provided for the phonemes of the annotation and the vertical axis being provided for the phonemes of the query, and showing a number of lattice points, each corresponding to a possible match between an annotation
20 phoneme and a query phoneme;

Figure 9a schematically illustrates the dynamic programming constraints employed in a dynamic programming matching process when the annotation is a typed input and
25 the query is generated from a voiced input;

Figure 9b schematically illustrates the dynamic programming constraints employed in a dynamic programming matching process when the query is a typed input and when
30 the annotation is a voiced input;

Figure 10 schematically illustrates the deletion and decoding probabilities which are stored for an example phoneme;

Figure 11 schematically illustrates the dynamic programming constraints employed in a dynamic programming matching process when both the annotation and the query are voiced inputs;

5

Figure 12 is a flow diagram illustrating the main processing steps performed in the dynamic programming matching process;

10

Figure 13 is a flow diagram illustrating the main processing steps employed to begin the dynamic programming process by propagating from a null start node to all possible start points;

15

Figure 14 is a flow diagram illustrating the main processing steps employed to propagate dynamic programming paths from the start points to all possible end points;

20

Figure 15 is a flow diagram illustrating the main processing steps employed in propagating the paths from the end points to a null end node;

25

Figure 16a is a flow diagram illustrating part of the processing steps performed in propagating a path using the dynamic programming constraints;

30

Figure 16b is a flow diagram illustrating the remaining process steps involved in propagating a path using the dynamic programming constraints;

Figure 17 is a flow diagram illustrating the processing steps involved in determining a transition score for propagating a path from a start point to an end point;

Figure 18a is a flow diagram illustrating part of the processing steps employed in calculating scores for deletions and decodings of annotation and query phonemes;

5 Figure 18b is a flow diagram illustrating the remaining steps employed in determining scores for deletions and decodings of annotation and query phonemes;

10 Figure 19a schematically illustrates an alternative embodiment which employs a different technique for aligning the query with each annotation;

15 Figure 19b is a plot illustrating the way in which a dynamic programming score varies with a comparison of a query with an annotation in the embodiment illustrated in Figure 19a;

20 Figure 20 is a schematic block diagram illustrating the form of an alternative user terminal which is operable to retrieve a data file from a database located within a remote server in response to an input voice query;

25 Figure 21 illustrates another user terminal which allows a user to retrieve data from a database located within a remote server in response to an input voice query;

30 Figure 22 is a schematic block diagram illustrating the form of a phoneme string and four M-grams generated from the phoneme string; and

Figure 23 is a plot illustrating the way in which a dynamic programming score varies with a comparison of one of the M-grams shown in Figure 22 with a sequence of annotation phonemes.

Embodiments of the present invention can be implemented using dedicated hardware circuits, but the embodiment to be described is implemented in computer software or code, which is run in conjunction with processing hardware such as a personal computer, workstation, photocopier, facsimile machine, personal digital assistant (PDA) or the like.

DATA FILE ANNOTATION

Figure 1 illustrates the form of a user terminal 59 which allows a user to input typed or voiced annotation data via the keyboard 3 and microphone 7 for annotating a data file 91 which is to be stored in a database 29. In this embodiment, the data file 91 comprises a two dimensional image generated by, for example, a camera. The user terminal 59 allows the user 39 to annotate the 2D image with an appropriate annotation which can be used subsequently for retrieving the 2D image from the database 29. In this embodiment, a typed input is converted, by the phonetic transcription unit 75, into phoneme (or phoneme-like) and word lattice annotation data which is passed to the control unit 55. Figure 2 illustrates the form of the phoneme and word lattice annotation data generated for the typed input "picture of the Taj Mahal". As shown in Figure 2, the phoneme and word lattice is an acyclic directed graph with a single entry point and a single exit point. It represents different parses of the user's input. As shown, the phonetic transcription unit 75 identifies a number of different possible phoneme strings which correspond to the typed input, from an internal phonetic dictionary (not shown).

Similarly, a voiced input is converted by the automatic

speech recognition unit 51 into phoneme (or phoneme-like) and word lattice annotation data which is also passed to the control unit 55. The automatic speech recognition unit 51 generates this phoneme and word lattice annotation data by (i) generating a phoneme lattice for the input utterance; (ii) then identifying words within the phoneme lattice; and (iii) finally by combining the two. Figure 3 illustrates the form of the phoneme and word lattice annotation data generated for the input utterance "picture of the Taj Mahal". As shown, the automatic speech recognition unit identifies a number of different possible phoneme strings which correspond to this input utterance. As is well known in the art of speech recognition, these different possibilities can have their own weighting which is generated by the speech recognition unit 51 and is indicative of the confidence of the speech recognition unit's output. In this embodiment, however, this weighting of the phonemes is not performed. As shown in Figure 3, the words which the automatic speech recognition unit 51 identifies within the phoneme lattice are incorporated into the phoneme lattice data structure. As shown for the example phrase given above, the automatic speech recognition unit 51 identifies the words "picture", "of", "off", "the", "other", "ta", "tar", "jam", "ah", "hal", "ha" and "al".

As shown in Figure 3, the phoneme and word lattice generated by the automatic speech recognition unit 51 is an acyclic directed graph with a single entry point and a single exit point. It represents different parses of the user's input annotation utterance. It is not simply a sequence of words with alternatives, since each word does not have to be replaced by a single alternative, one word can be substituted for two or more words or

phonemes, and the whole structure can form a substitution for one or more words or phonemes. Therefore, the density of the data within the phoneme and word lattice annotation data essentially remains linear throughout the annotation data, rather than growing exponentially as in the case of a system which generates the N-best word lists for the audio annotation input.

In this embodiment, the annotation data generated by the automatic speech recognition unit 51 or the phonetic transcription unit 75 has the following general form:

HEADER

- flag if word if phoneme if mixed
- time index associating the location of blocks of annotation data within memory to a given time point.
- word set used (i.e. the dictionary)
- phoneme set used
- the language to which the vocabulary pertains
- phoneme probability data

Block(i) $i = 0, 1, 2, \dots$

node N_j $j = 0, 1, 2, \dots$

- time offset of node from start of block
- phoneme links (k) $k = 0, 1, 2, \dots$
offset to node $N_j = N_k - N_j$ (N_k is node to which link K extends)
phoneme associated with link (k)
- word links (l) $l = 0, 1, 2, \dots$
offset to node $N_j = N_l - N_j$ (N_l is node to which link l extends)
word associated with link (l)

The flag identifying if the annotation data is word

annotation data, phoneme annotation data or if it is mixed is provided since not all the data files within the database will include the combined phoneme and word lattice annotation data discussed above, and in this case, a different search strategy would be used to search this annotation data.

In this embodiment, the annotation data is divided into blocks of nodes in order to allow the search to jump into the middle of the annotation data for a given search. The header therefore includes a time index which associates the location of the blocks of annotation data within the memory to a given time offset between the time of start and the time corresponding to the beginning of the block.

The header also includes data defining the word set used (i.e. the dictionary), the phoneme set used and their probabilities and the language to which the vocabulary pertains. The header may also include details of the automatic speech recognition system used to generate the annotation data and any appropriate settings thereof which were used during the generation of the annotation data.

The blocks of annotation data then follow the header and identify, for each node in the block, the time offset of the node from the start of the block, the phoneme links which connect that node to other nodes by phonemes and word links which connect that node to other nodes by words. Each phoneme link and word link identifies the phoneme or word which is associated with the link. They also identify the offset to the current node. For example, if node N_{50} is linked to node N_{55} by a phoneme

link, then the offset to node N_{50} is 5. As those skilled in the art will appreciate, using an offset indication like this allows the division of the continuous annotation data into separate blocks.

5

In an embodiment where an automatic speech recognition unit outputs weightings indicative of the confidence of the speech recognition units output, these weightings or confidence scores would also be included within the data structure. In particular, a confidence score would be provided for each node which is indicative of the confidence of arriving at the node and each of the phoneme and word links would include a transition score depending upon the weighting given to the corresponding phoneme or word. These weightings would then be used to control the search and retrieval of the data files by discarding those matches which have a low confidence score.

20

In response to the user's input, the control unit 55 retrieves the appropriate 2D file from the database 29 and appends the generated phoneme and word annotation data to the data file 91. The augmented data file is then returned to the database 29. During this annotating step, the control unit 55 is operable to display the 2D image on the display 57, so that the user can ensure that the annotation data is associated with the correct data file 91.

25

30

As will be explained in more detail below, the use of such phoneme and word lattice annotation data allows a quick and efficient search of the database 29 to be carried out, to identify and to retrieve a desired 2D image data file stored therein. This can be achieved by

firstly searching in the database 29 using the word data and, if this search fails to provide the required data file, then performing a further search using the more robust phoneme data. As those skilled in the art of speech recognition will realise, use of phoneme data is more robust because phonemes are dictionary independent and allow the system to cope with out of vocabulary words, such as names, places, foreign words etc. Use of phoneme data is also capable of making the system future-proof, since it allows data files which are placed into the database 29 to be retrieved when the original annotation was input by voice and the original automatic speech recognition system did not understand the words of the input annotation.

DATA FILE RETRIEVAL

Figure 4 is a block diagram illustrating the form of a user terminal 59 which is used, in this embodiment, to retrieve the annotated 2D images from the database 29. This user terminal 59 may be, for example, a personal computer, a hand-held device or the like. As shown, in this embodiment, the user terminal 59 comprises the database 29 of annotated 2D images, an automatic speech recognition unit 51, a phonetic transcription unit 75, a keyboard 3, a microphone 7, a search engine 53, a control unit 55 and a display 57. In operation, the user inputs either a voice query via the microphone 7 or a typed query via the keyboard 3 and the query is processed either by the automatic speech recognition unit 51 or the phonetic transcription unit 75 to generate corresponding phoneme and word data. This data may also take the form of a phoneme and word lattice, but this is not essential. This phoneme and word data is then input to the control unit 55 which is operable to initiate an appropriate

search of the database 29 using the search engine 53. The results of the search, generated by the search engine 53, are then transmitted back to the control unit 55 which analyses the search results and generates and displays appropriate display data (such as the retrieved 2D image) to the user via the display 57.

Figures 5a and 5b are flow diagrams which illustrate the way in which the user terminal 59 operates in this embodiment. In step s1, the user terminal 59 is in an idle state and awaits an input query from the user 39. Upon receipt of an input query, the phoneme and word data for the input query is generated in step s3 by the automatic speech recognition unit 51 or the phonetic transcription unit 75. The control unit 55 then instructs the search engine 53, in step s5, to perform a search in the database 29 using the word data generated from the input query. The word search employed in this embodiment is the same as is currently being used in the art for typed word searches, and will not be described in more detail here. If in step s7, the control unit 55 identifies from the search results, that a match for the user's input query has been found, then it outputs the search results to the user via the display 57.

In this embodiment, the user terminal 59 then allows the user to consider the search results and awaits the user's confirmation as to whether or not the results correspond to the information the user requires. If they are, then the processing proceeds from step s11 to the end of the processing and the user terminal 59 returns to its idle state and awaits the next input query. If, however, the user indicates (by, for example, inputting an appropriate voice command) that the search results do not correspond

to the desired information, then the processing proceeds from step s11 to step s13, where the search engine 53 performs a phoneme search of the database 29. However, in this embodiment, the phoneme search performed in step
5 s13 is not of the whole database 29, since this could take several hours depending on the size of the database 29.

Instead, the phoneme search performed in step s13 uses
10 the results of the word search performed in step s5 to identify one or more portions within the database which may correspond to the user's input query. For example, if the query comprises three words and the word search only identifies one or two of the query words in the
15 annotation, then it performs a phoneme search of the portions of the annotations surrounding the identified word or words. The way in which the phoneme search performed in step s13 is carried out in this embodiment will be described in more detail later.

20 After the phoneme search has been performed, the control unit 55 identifies, in step s15, if a match has been found. If a match has been found, then the processing proceeds to step s17 where the control unit 55 causes the
25 search results to be displayed to the user on the display 57. Again, the system then awaits the user's confirmation as to whether or not the search results correspond to the desired information. If the results are correct, then the processing passes from step s19 to
30 the end and the user terminal 59 returns to its idle state and awaits the next input query. If however, the user indicates that the search results do not correspond to the desired information, then the processing proceeds from step s19 to step s21, where the control unit 55 is

operable to ask the user, via the display 57, whether or not a phoneme search should be performed of the whole database 29. If in response to this query, the user indicates that such a search should be performed, then
5 the processing proceeds to step s23 where the search engine performs a phoneme search of the entire database 29.

On completion of this search, the control unit 55
10 identifies, in step s25, whether or not a match for the user's input query has been found. If a match is found, then the processing proceeds to step s27 where the control unit 55 causes the search results to be displayed to the user on the display 57. If the search results are
15 correct, then the processing proceeds from step s29 to the end of the processing and the user terminal 59 returns to its idle state and awaits the next input query. If, on the other hand, the user indicates that the search results still do not correspond to the desired
20 information, then the processing passes to step s31 where the control unit 55 queries the user, via the display 57, whether or not the user wishes to redefine or amend the search query. If the user does wish to redefine or amend the search query, then the processing returns to step s3
25 where the user's subsequent input query is processed in a similar manner. If the search is not to be redefined or amended, then the search results and the user's initial input query are discarded and the user terminal 59 returns to its idle state and awaits the next input
30 query.

A general description has been given above of the way in which a search is carried out in this embodiment by the user terminal 59. A more detailed description will now

be given of the way in which the search engine 53 carries out the phoneme searches, together with a brief description of the motivation underlying the search strategy.

5

INFORMATION RETRIEVAL AS A CLASSIFICATION PROBLEM

In the classic classification scenario, a test datum has to be classified into one of K classes. This is done using knowledge about other data for which the class is known. The classification problem assumes that there is a "class" random variable which can take values from 1 to K . The optimal classification then being found by identifying the class to which the test datum most likely belongs. It is assumed that the training data is generated by N generative processes which resulted in n_k data of class k , where $\sum_{k=1}^K n_k = N$. Denoting the vector (n_1, n_2, \dots, n_K) by n , the training data by D and the test datum by x , the classic classification problem is to determine the value of k which maximises the following probability:

20

$$P(k|x,D,n) = \frac{P(x|k,D,n)P(k|D,n)}{P(x|D)} \quad (1)$$

25

30

The second term on the numerator is a prior probability for the class which gives more weight to classes which occur more often. In the context of information retrieval, each class has a single training datum (i.e. the annotation data). Therefore, for information retrieval, the second term on the right-hand side of the above expression can be ignored. Similarly, the denominator can also be ignored since $P(x|D)$ is the same for each class and therefore just normalises the

numerator. Consequently, the order of the classes can be ranked by simply ranking the order of the first term on the numerator of the above expression for the classes. In other words, determining and ranking $P(x|d_k)$ for all the classes, where d_k is the training datum for class k .

In this embodiment, the test datum x represents the input query and the training datum for class k (i.e. d_k) represents the k^{th} annotation, and it is assumed that there is an underlying statistical model (M) which generated both the query and the annotation, as illustrated in Figure 6a. In the general case, this model has three unknowns: the model structure, m , the state sequences through that model for both the query and the annotation, s_q and s_a , and the output distribution C . In this case, we know the output distribution since it embodies the characteristics of the speech recognition system which generates the phoneme strings from the input speech. As will be described later, it can be obtained by applying a large database of known speech to the speech recognition system, and it will be referred to hereinafter as the confusion statistics. Therefore, introducing the state sequences and the model into the above probabilities (and using the variables q for the input query and a for the annotation) yields:

$$P(q|a) = \sum_m \sum_{s_a} \sum_{s_q} P(q|m, s_q, s_a, C, a) P(m, s_q, s_a | C, a) \quad (2)$$

which can be expanded using Bayesian methods to give:

$P(q|a)$

$$= \frac{\sum_m \sum_{s_a} \sum_{s_q} P(q|m, s_q, C) P(a|m, s_a, C) P(s_q|m, C) P(s_a|m, C) P(m|C)}{\sum_m \sum_{s_a} \sum_{s_q} P(a|m, s_a, C) P(s_a|m, C) P(m|C)} \quad (3)$$

Although the above expression looks complicated, the summations over the set of state sequences s_q and s_a can be performed using a standard dynamic programming algorithm. Further, the last term on both the numerator and the denominator can be ignored, since it can be assumed that each model is equally likely and the state sequence terms $P(s|m, c)$ can be ignored because it can also be assumed that each state sequence is equally likely. Further, by assuming that the underlying model structure is a canonical sequence of phonemes having approximately the same length as the query, subject to insertions, the summation over the different models can be removed, although it is replaced with a summation over all possible phonemes because, in the general case, the canonical sequence of phonemes of the model is unknown. Therefore, ignoring the state sequence summations, the term which is to be evaluated inside the dynamic programming algorithm becomes:

$$\sum_{r=1}^{N_p} P(a_i|p_r, C) P(q_j|p_r, C) P(p_r|C) \quad (4)$$

on the numerator, and

$$\sum_{r=1}^{N_p} P(a_i|p_r, C)P(p_r|C) \quad (5)$$

on the denominator (i.e. the normalising term), where N_p is the total number of phonemes known to the system and a_i , q_j and p_r are the annotation phoneme, query phoneme and model phoneme respectively which correspond to the current DP lattice point being evaluated. As can be seen from a comparison of equations (4) and (5), the probability terms calculated on the denominator are calculated on the numerator as well. Therefore, both terms can be accumulated during the same dynamic programming routine. Considering the probabilities which are determined in more detail, $P(q_j|p_r, C)$ is the probability of decoding canonical phoneme p_r as query phoneme q_j given the confusion statistics; $P(a_i|p_r, C)$ is the probability of decoding canonical phoneme p_r as annotation phoneme a_i given the confusion statistics; and $P(p_r|C)$ is the probability of canonical phoneme p_r occurring unconditionally given the confusion statistics.

In addition to the above terms, at each point in the dynamic programming calculation, a further term must be calculated which deals with insertions and deletions in the query or the annotation relative to the model. As those skilled in the art will appreciate, an insertion or a deletion in the query is independent from an insertion or a deletion in the annotation and vice versa. Therefore, these additional terms are dealt with separately. Insertions and deletions in the annotation relative to the model must also be considered for the

normalising term given in equation (5) above.

As those skilled in the art will appreciate from the description of Figures 4 and 5, in this embodiment, the annotation phoneme data and the query phoneme data may both be derived either from text or speech. Therefore, there are four situations to consider:

- i) both the annotation and the query are generated from text;
- ii) the annotation is generated from text and the query is generated from speech;
- iii) the annotation is generated from speech and the query is generated from text; and
- iv) both the query and annotation are generated from speech.

The first situation is the simple case in which there can be no time compression/expansion of the annotation or the query and the comparison between the annotation and the query is performed by a simple boolean comparison of the respective phoneme sequences.

In the second situation, the annotation is taken to be correct and the dynamic programming alignment allows the insertion and deletion of phonemes in the query in order to find the best alignment between the two. To illustrate this case, Figure 6b shows a possible matching between a sequence of annotation phonemes (labelled a_0 , a_1 , a_2 ...) and a sequence of query phonemes (labelled q_0 , q_1 , q_2 ...), when the annotation phonemes are generated from text. As illustrated by the dashed arrows, annotation phoneme a_0 is aligned with query phoneme q_0 , annotation phoneme a_1 is aligned with query phoneme q_2 ,

annotation phoneme a_2 is aligned with query phoneme q_3 ,
 annotation phoneme a_3 is aligned with query phoneme q_3 ,
 and annotation phoneme a_4 is aligned with query phoneme
 q_4 . For each of these alignments, the dynamic
 programming routine calculates the terms given in
 equations (4) and (5) above. However, in this case,
 these equations simplify because the canonical sequence
 of model phonemes is known (since these are the
 annotation phonemes). In particular, the normalising
 term is one because the annotation is the model and the
 numerator simplifies to $P(q_i|a_j, C)$. In addition to these
 decoding terms, the dynamic programming routine also
 calculates the relevant insertion and deletion
 probabilities for the phonemes which are inserted in the
 query relative to the annotation (such as query phoneme
 q_1) and for the phonemes which are deleted in the query
 relative to the annotation (represented by query phoneme
 q_3 which is matched with the two annotation phonemes a_2
 and a_3).

The third situation mentioned above is similar to the
 second situation except the sequence of query phonemes is
 taken to be correct and the dynamic programming alignment
 allows the insertion and deletion of phonemes in the
 annotation relative to the query. However, in this
 situation, equations (1) to (5) cannot be used because
 the query is known. Therefore, in this situation,
 equation (1) can be reformulated as:

$$P(K|x, D, n) = \frac{P(D|K, x, n)P(K|x, n)}{P(D|x)} \quad (6)$$

As with the corresponding terms in the equation (1)

above, the second term on the numerator and the denominator can both be ignored. The first term of the numerator in equation (6) above can be expanded in a similar way to the way in which the first term on the numerator of equation (1) was expanded. However, in this situation, with the query being taken to be the model, the normalising term calculated during the dynamic programming routine simplifies to one and the numerator simplifies to $P(a_i|q_j, C)$. Like the second situation discussed above, the dynamic programming routine also calculates the relevant insertion and deletion probabilities for the phonemes which are inserted in the annotation relative to the query and for the phonemes which are deleted in the annotation relative to the query.

Finally, in the fourth situation, when both the annotation and the query are generated from speech, both sequences of phoneme data can have insertions and deletions relative to the unknown canonical sequence of model phonemes which represents the text of what was actually spoken. This is illustrated in Figure 6c, which shows a possible matching between a sequence of annotation phonemes (labelled $a_i, a_{i+1}, a_{i+2} \dots$), a sequence of query phonemes (labelled $q_j, q_{j+1}, q_{j+2} \dots$) and a sequence of phonemes (labelled $p_n, p_{n+1}, p_{n+2} \dots$) which represents the canonical sequence of phonemes of what was actually spoken by both the query and the annotation. As shown in Figure 6c, in this case, the dynamic programming alignment technique must allow for the insertion of phonemes in both the annotation and the query (represented by the inserted phonemes a_{i+3} and q_{j+1}) as well as the deletion of phonemes from both the annotation and the query (represented by phonemes a_{i+1} and

q_{j+2} , which are both aligned with two phonemes in the canonical sequence of phonemes), relative to the canonical sequence of model phonemes.

5 As those skilled in the art will appreciate, by introducing the model sequence of phonemes into the calculations, the algorithm is more flexible to pronunciation variations in both the query and the annotation.

10

A general description has been given above of the way in which the present embodiment performs information retrieval by matching the sequence of query phonemes with the sequences of annotation phonemes in the database. In
15 order to understand the operation of the present embodiment further, a brief description will now be given of a standard dynamic programming algorithm followed by a more detailed description of the particular algorithm used in this embodiment.

20

OVERVIEW OF DP SEARCH

As those skilled in the art know, dynamic programming is a technique which can be used to find the optimum alignment between sequences of features, which in this
25 embodiment are phonemes. It does this by simultaneously propagating a plurality of dynamic programming paths, each of which represents a possible matching between a sequence of annotation phonemes and a sequence of query phonemes. All paths begin at a start null node, which is
30 at the beginning of the annotation and the query, and propagate until they reach an end null node, which is at the end of the annotation and the query. Figures 7 and 8 schematically illustrate the matching which is performed and this path propagation. In particular,

Figure 7 shows a rectangular coordinate plot with the horizontal axis being provided for the annotation and the vertical axis being provided for the query. The start null node \emptyset_s is provided in the top left hand corner and the end null node \emptyset_e is provided on the bottom right hand corner. As shown in Figure 8, the phonemes of the annotation are provided along the horizontal axis and the phonemes of the query are provided along the vertical axis. Figure 8 also shows a number of lattice points, each of which represents a possible alignment between a phoneme of the annotation and a phoneme of the query. For example, lattice point 21 represents a possible alignment between annotation phoneme a_3 and query phoneme q_1 . Figure 8 also shows three dynamic programming paths m_1 , m_2 and m_3 which represent three possible matchings between the sequences of phonemes representative of the annotation and of the query and which begin at the start null node \emptyset_s and propagate through the lattice points to the end null node \emptyset_e . Referring back to equations (2) and (3) above, these dynamic programming paths represent the different state sequences s_q and s_a discussed above.

As represented by the different lengths of the horizontal and vertical axes shown in Figure 7, the input query does not need to include all the words of the annotation. For example, if the annotation is "picture of the Taj Mahal", then the user can simply search the database 29 for this picture by inputting the query "Taj Mahal". In this situation, the optimum alignment path would pass along the top horizontal axis until the query started to match the annotation. It would then start to pass through the lattice points to the lower horizontal axis and would end at the end node. This is illustrated in Figure 7 by the path 23. However, as those skilled in the art will

appreciate, the words in the query must be in the same order as they appear in the annotation, otherwise the dynamic programming alignment will not work.

5 In order to determine the similarity between the sequence
of annotation phonemes and the sequence of query
phonemes, the dynamic programming process keeps a score
for each of the dynamic programming paths which it
propagates, which score is dependent upon the overall
10 similarity of the phonemes which are aligned along the
path. In order to limit the number of deletions and
insertions of phonemes in the sequences being matched,
the dynamic programming process places certain
constraints on the way in which the dynamic programming
15 paths can propagate. As those skilled in the art will
appreciate, these dynamic programming constraints will be
different for the four situations discussed above.

DP CONSTRAINTS

20 *Both annotation and query are text.*
In the case where the query phoneme data and the
annotation phoneme data are both generated from text, the
dynamic programming alignment degenerates into a boolean
match between the two phoneme sequences and no phoneme
25 deletions or insertions are allowed.

Annotation is text and query is speech.

In the case where the annotation phoneme data is
generated from text and the query phoneme data is
30 generated from speech, there can be no phoneme deletions
or insertions in the annotation but there can be phoneme
deletions and insertions in the query relative to the
annotation. Figure 9a illustrates the dynamic
programming constraints which are used in this

embodiment, when the annotation is generated from text and the query is generated from speech. As shown, if a dynamic programming path ends at lattice point (i,j) , representing an alignment between annotation phoneme a_i and query phoneme q_j , then that dynamic programming path can propagate to the lattice points $(i+1,j)$, $(i+1,j+1)$ and $(i+1,j+2)$. A propagation to point $(i+1,j)$ represents the case when there is a deletion of a phoneme from the spoken query as compared with the typed annotation; a propagation to the point $(i+1,j+1)$ represents the situation when there is a simple decoding between the next phoneme in the annotation and the next phoneme in the query; and a propagation to the point $(i+1,j+2)$ represents the situation when there is an insertion of phoneme q_{j+1} in the spoken query as compared with the typed annotation and when there is a decoding between annotation phoneme a_{i+1} and query phoneme q_{j+2} .

Annotation is speech and query is text.

In the case where the annotation is generated from speech and the query is generated from text, there can be no insertions or deletions of phonemes from the query but there can be insertions and deletions from the annotation relative to the query. Figure 9b illustrates the dynamic programming constraints which are used in this embodiment, when the annotation is generated from speech and the query is generated from text. As shown, if a dynamic programming path ends at lattice point (i,j) , representing an alignment between annotation phoneme a_i and query phoneme q_j , then that dynamic programming path can propagate to the lattice points $(i,j+1)$, $(i+1,j+1)$ and $(i+2,j+1)$. A propagation to point $(i,j+1)$ represents the case when there is a deletion of a phoneme from the spoken annotation as compared with the typed query; a

propagation to the point $(i+1, j+1)$ represents the situation when there is a simple decoding between the next phoneme in the annotation and the next phoneme in the query; and a propagation to the point $(i+2, j+1)$ represents the situation when there is an insertion of phoneme a_{i+1} in the spoken annotation as compared with the typed query and when there is a decoding between annotation phoneme a_{i+2} and query phoneme q_{i+1} .

10 *Annotation is speech and query is speech.*

In the case where both the annotation and the query are generated from speech, phonemes can be inserted and deleted from each of the annotation and the query relative to the other. Figure 11 shows the dynamic programming constraints which are used in this embodiment, when both the annotation phonemes and the query phonemes are generated from speech. In particular, if a dynamic programming path ends at lattice point (i, j) , representing an alignment between annotation phoneme a_i and query phoneme q_j , then that dynamic programming path can propagate to the lattice points $(i+1, j)$, $(i+2, j)$, $(i+3, j)$, $(i, j+1)$, $(i+1, j+1)$, $(i+2, j+1)$, $(i, j+2)$, $(i+1, j+2)$ and $(i, j+3)$. These propagations therefore allow the insertion and deletion of phonemes in both the annotation and the query relative to the unknown canonical sequence of model phonemes corresponding to the text of what was actually spoken.

Beginning and End DP Constraints

30 In this embodiment, the dynamic programming alignment operation allows a dynamic programming path to start and end at any of the annotation phonemes. As a result, the query does not need to include all the words of the annotation, although the query words do need to be in the

same order as they appear in the annotation.

DP SCORE PROPAGATION

As mentioned above, the dynamic programming process keeps
5 a score for each of the dynamic programming paths, which
score is dependent upon the similarity of the phonemes
which are aligned along the path. Therefore, when
propagating a path ending at point (i,j) to these other
points, the dynamic programming process adds the
10 respective "cost" of doing so to the cumulative score for
the path ending at point (i,j), which is stored in a
store (SCORE(i,j)) associated with that point. As those
skilled in the art will appreciate, this cost includes
the above-mentioned insertion probabilities, deletion
15 probabilities and decoding probabilities. In particular,
when there is an insertion, the cumulative score is
multiplied by the probability of inserting the given
phoneme; when there is a deletion, the cumulative score
is multiplied by the probability of deleting the phoneme;
20 and when there is a decoding, the cumulative score is
multiplied by the probability of decoding the two
phonemes.

In order to be able to calculate these probabilities, the
25 system stores a probability for all possible phoneme
combinations. In this embodiment, the deletion of a
phoneme in the annotation or the query is treated in a
similar manner to a decoding. This is achieved by simply
treating a deletion as another phoneme. Therefore, if
30 there are 43 phonemes known to the system, then the
system will store one thousand eight hundred and ninety
two ($1892 = 43 \times 44$) decoding/deletion probabilities, one
for each possible phoneme decoding and deletion. This is
illustrated in Figure 10, which shows the possible

phoneme decodings which are stored for the phoneme /ax/ and which includes the deletion phoneme (\emptyset) as one of the possibilities. As those skilled in the art will appreciate, all the decoding probabilities for a given phoneme must sum to one, since there are no other possibilities. In addition to these decoding/deletion probabilities, the system stores 43 insertion probabilities, one for each possible phoneme insertion. As will be described later, these probabilities are determined in advance from training data.

To illustrate the score propagations, a number of examples will now be considered. In the case where the annotation is text and the query is speech and for the path propagating from point (i,j) to point $(i+1,j+2)$, the phoneme q_{j+1} is inserted relative to the annotation and query phoneme q_{j+2} is decoded with annotation phoneme a_{i+1} . Therefore, the score propagated to point $(i+1,j+2)$ is given by:

$$S(i+1,j+2) = S(i,j) \cdot PI(q_{j+1}|C) \cdot P(q_{j+2}|a_{i+1},C)$$

(7)

where $PI(q_{j+1}|C)$ is the probability of inserting phoneme q_{j+1} in the spoken query and $P(q_{j+2}|a_{i+1},C)$ represents the probability of decoding annotation phoneme a_{i+1} as query phoneme q_{j+2} .

In the case where both the annotation and the query are generated from speech and when propagating from point (i,j) to point $(i+2,j+1)$, the annotation phoneme a_{i+1} is inserted relative to the query and there is a decoding between annotation phoneme a_{i+2} and query phoneme q_{j+1} .

Therefore, the score propagated to point $(i+2, j+1)$ is given by:

$$S(i+2, j+1) = S(i, j) \cdot P(a_{i+1} | C) \cdot \sum_{r=1}^{N_p} P(a_{i+2} | p_r C) P(q_{j+1} | p_r C) P(p_r | C)$$

(8)

As those skilled in the art will appreciate, during this path propagation, several paths will meet at the same lattice point. In this embodiment, the scores associated with the paths which meet are simply added together. Alternatively, a comparison between the scores could be made and the path having the best score could be continued whilst the other path(s) is (are) discarded. However, this is not essential in this embodiment, since the dynamic programming process is only interested in finding a score which represents the similarity between the phoneme data of the query and the phoneme data of the annotation. It is not interested in knowing what the best alignment between the two is.

If both the query and the annotation are generated from speech, then once all the paths have been propagated to the end node \emptyset , and a total score for the similarity between the query and the current annotation has been determined, the system normalises this score using the normalising term which has been accumulating during the DP process. The system then compares the query with the next annotation in a similar manner. Once the query has been matched with all the annotations, the normalised scores for the annotations are ranked and based on the ranking, the system outputs to the user the annotation(s) most similar to the input query.

DETAILED DESCRIPTION OF DP SEARCH

A more detailed description will now be given of the way in which the dynamic programming search is carried out when matching a sequence of query phonemes with a sequence of annotation phonemes. Referring to Figure 12, in step s101, the system initialises the dynamic programming scores. Then in step s103, the system propagates paths from the null start node (\emptyset_s) to all possible start points. Then in step s105, the system propagates the dynamic programming paths from all the start points to all possible end points using the dynamic programming constraints discussed above. Finally in step s107, the system propagates the paths ending at the end points to the null end node (\emptyset_e).

Figure 13 shows in more detail, the processing steps involved in step s103 in propagating the dynamic programming paths from the null start node (\emptyset_s) to all the possible start points, which are defined by the dynamic programming constraints. One of the constraints is that a dynamic programming path can begin at any of the annotation phonemes and the other constraint, which defines the number of hops allowed in the sequence of query phonemes, depends upon whether or not the query is text or speech. In particular, if the query is generated from text, then the start points comprise the first row of lattice points in the search space, i.e. points $(i,0)$ for $i = 0$ to $N_{ann}-1$; and if the query is generated from speech, then the start points comprise the first four rows of lattice points in the search space, i.e. points $(i,0)$, $(i,1)$, $(i,2)$ and $(i,3)$ for $i = 0$ to $N_{ann}-1$.

The way in which this is achieved will now be described with reference to the steps shown in Figure 13. As

shown, in step s111, the system determines whether or not the input query is a text query. If it is, then the processing proceeds to step s113 where the system sets the value of the variable mx to one, which defines the maximum number of "hops" allowed in the sequence of query phonemes when the query is text. The processing then proceeds to steps s115, s117 and s119 which are operable to start a dynamic programming path at each of the lattice points in the first row of the search space, by adding the transition score for passing from the null start node to the lattice point (i,0) to the score (SCORE(i,0)) associated with point (i,0), for i = 0 to Nann-1. When the query is text, this ends the processing in step s103 shown in Figure 12 and the processing then proceeds to step s105.

If the system determines at step s111, that the query is not text and was therefore generated from a spoken input, the system proceeds to step s121 where mx is set to mxhops which is a constant having a value which is one more than the maximum number of "hops" allowed by the dynamic programming constraints. As shown in Figures 9 and 10, in the case where the query is speech, a path can jump at most to a query phoneme which is three phonemes further along the sequence of query phonemes. Therefore, in this embodiment, mxhops has a value of four and the variable mx is set equal to four, provided there are four or more phonemes in the query, otherwise mx is set equal to the number of phonemes in the query. The processing then proceeds to steps s123, s125, s127, s129 and s131 which are operable to begin dynamic programming paths at each of the lattice points in the first four rows of the search space by adding the corresponding transition probability to the score associated with the

corresponding lattice point. When the query is generated from a spoken input, this ends the processing in step s103 shown in Figure 12 and the processing then proceeds to step s105.

5

In this embodiment, the system propagates the dynamic programming paths from the start points to the end points in step s105 by processing the lattice points in the search space column by column in a raster like technique. The control algorithm used to control this raster processing operation is shown in Figure 14. In step s151, the system compares an annotation phoneme loop pointer i with the number of phonemes in the annotation (Nann). Initially the annotation phoneme loop pointer i is set to zero and the processing will therefore initially proceed to step s153 where a similar comparison is made for a query phoneme loop pointer j relative to the total number of phonemes in the query (Nquery). Initially the loop pointer j is also set to zero and therefore the processing proceeds to step s155 where the system propagates the path ending at point (i, j) using the dynamic programming constraints discussed above. The way in which the system propagates the paths in step s155 will be described in more detail later. After step s155, the loop pointer j is incremented by one in step s157 and the processing returns to step s153. Once this processing has looped through all the phonemes in the query (thereby processing the current column of lattice points), the processing proceeds to step s159 where the query phoneme loop pointer j is reset to zero and the annotation phoneme loop pointer i is incremented by one. The processing then returns to step s151 where a similar procedure is performed for the next column of lattice points. Once the last column of lattice points have been

10

15

20

25

30

processed, the processing proceeds to step s161 where the annotation phoneme loop pointer i is reset to zero and the processing in step s105 shown in Figure 12 ends.

5 Figure 15 shows in more detail the processing steps involved in step s107 shown in Figure 12, when propagating the paths at the end points to the end null node \emptyset_e . As with the propagation from the start null node \emptyset_s , the lattice points which are the "end points",
 10 are defined by the dynamic programming constraints, which depend upon whether the query is text or speech. Further, in this embodiment, the dynamic programming constraints allow dynamic programming paths to exit the annotation at any point along the sequence of annotation
 15 phonemes. Therefore, if the query is text, then the system will allow dynamic programming paths ending in the last row of the lattice points, i.e. at points $(i, N_{\text{query}}-1)$ for $i = 0$ to $N_{\text{ann}}-1$, to propagate to the end null node \emptyset_e . If, however, the query was generated from speech,
 20 then the system allows any path propagating in the last four rows of the lattice points, i.e. points $(i, N_{\text{query}}-4)$, $(i, N_{\text{query}}-3)$, $(i, N_{\text{query}}-2)$ and $(i, N_{\text{query}}-1)$ for $i = 0$ to $N_{\text{ann}}-1$, to propagate to the end null node \emptyset_e .

25 As shown in Figure 15, this process begins at step s171 where the system determines whether or not the query is text. If it is, then the processing proceeds to step s173 where the query phoneme loop pointer j is set to $N_{\text{query}}-1$. The processing then proceeds to step s175
 30 where the annotation phoneme loop pointer i is compared with the number of phonemes in the annotation (N_{ann}). Initially the annotation phoneme loop pointer i is set to zero and therefore the processing will proceed to step s177 where the system calculates the transition score

from point (i,Nquery-1) to the null end node \emptyset_e . This transition score is then combined with the cumulative score for the path ending at point (i,Nquery-1) which is stored in SCORE(i,Nquery-1). As mentioned above, in this embodiment, the transition and cumulative scores are probability based and they are combined by multiplying the probabilities together. However, in this embodiment, in order to remove the need to perform multiplications and in order to avoid the use of high floating point precision, the system employs log probabilities for the transition and cumulative scores. Therefore, in step s179, the system adds the cumulative score for the path ending at point (i,Nquery-1) to the transition score calculated in step s177 and the result is copied to a temporary store, TEMPENDSCORE.

As mentioned above, if two or more dynamic programming paths meet at the same point, then the cumulative scores for each of the paths are added together. Therefore, since log probabilities are being used, the scores associated with paths which meet are effectively converted back to probabilities, added and then reconverted to log probabilities. In this embodiment, this operation is referred to as a "log addition" operation. This is a well known technique and is described in, for example, the book entitled "Automatic Speech Recognition. The Development of the (Sphinx) System" by Lee, Kai-Fu published by Kluwer Academic Publishers, 1989, at pages 28 and 29.

Since the path propagating from point (i,Nquery-1) to the null end node will meet with other dynamic programming paths, the system performs a log addition of TEMPENDSCORE with the score stored in the end node (ENDSCORE) and the

result is stored in ENDScore. The processing then proceeds to step s183 where the annotation phoneme loop pointer i is incremented. The processing then returns to step s175 where a similar process is performed for the
 5 next lattice point in the last row of lattice points. Once all the lattice points in the last row have been processed in this way, the processing performed in step s107 shown in Figure 12 ends.

10 If the system determines at step s171 that the query is not text, then the processing proceeds to step s185 where the query phoneme loop pointer j is set to the number of phonemes in the query minus $mxhops$, i.e. $N_{query}-4$. The processing then proceeds to step s187, where the system
 15 checks to see if the annotation phoneme loop pointer i is less than the number of phonemes in the annotation (N_{ann}). Initially the annotation phoneme loop pointer i is set to zero and therefore the processing proceeds to step s189 where the system checks to see if the query
 20 phoneme loop pointer j is less than the number of phonemes in the query (N_{query}). Initially it will be, and the processing proceeds to step s191 where the system calculates the transition score from lattice point (i,j) to the null end node \emptyset . This transition score is then
 25 added, in step s193, to the cumulative score for the path ending at point (i,j) and the result is copied to the temporary score, $TEMPENDSCORE$. The processing then proceeds to step s195 where the system performs a log addition of $TEMPENDSCORE$ with $ENDSCORE$ and the result is
 30 stored in $ENDSCORE$. The processing then proceeds to step s197 where the query phoneme loop pointer j is incremented by one and the processing returns to step s189. The above processing steps are then repeated until the query phoneme loop pointer j has been incremented so

that it equals the number of phonemes in the query (Nquery). The processing then proceeds to step s199, where the query phoneme loop pointer j is reset to Nquery-4 and the annotation phoneme loop pointer i is incremented by one. The processing then returns to step s187. The above processing steps are then repeated until all the lattice points in the last four rows of the search space have been processed in this way, after which the processing performed in step s107 shown in Figure 12 ends.

Propagate

In step s155 shown in Figure 14, the system propagates the path ending at lattice point (i,j) using the dynamic programming constraints discussed above. Figure 16 is a flowchart which illustrates the processing steps involved in performing this propagation step. As shown, in step s211, the system sets the values of two variables mxj and mxj and initialises annotation phoneme loop pointer i2 and query phoneme loop pointer j2. The loop pointers i2 and j2 are provided to loop through all the lattice points to which the path ending at point (i,j) can propagate to and the variables mxj and mxj are used to ensure that i2 and j2 can only take the values which are allowed by the dynamic programming constraints. In particular, mxj is set equal to i plus mxhops, provided this is less than or equal to the number of phonemes in the annotation, otherwise mxj is set equal to the number of phonemes in the annotation (Nann). Similarly, mxj is set equal to j plus mxhops, provided this is less than or equal to the number of phonemes in the query, otherwise mxj is set equal to the number of phonemes in the query (Nquery). Finally, in step s211, the system initialises the annotation phoneme loop pointer i2 to be equal to the

current value of the annotation phoneme loop pointer i and the query phoneme loop pointer $j2$ to be equal to the current value of the query phoneme loop pointer j .

5 Since the dynamic programming constraints employed by the system depend upon whether the annotation is text or speech and whether the query is text or speech, the next step is to determine how the annotation and the query were generated. This is performed by the decision blocks
 10 s213, s215 and s217. If both the annotation and the query are generated from speech, then the dynamic programming path ending at lattice point (i,j) can propagate to the other points shown in Figure 11 and process steps s219 to s235 operate to propagate this path
 15 to these other points. In particular, in step s219, the system compares the annotation phoneme loop pointer $i2$ with the variable mx_i . Since annotation phoneme loop pointer $i2$ is set to i and mx_i is set equal to $i+4$, in step s211, the processing will proceed to step s221 where
 20 a similar comparison is made for the query phoneme loop pointer $j2$. The processing then proceeds to step s223 which ensures that the path does not stay at the same lattice point (i,j) since initially, $i2$ will equal i and $j2$ will equal j . Therefore, the processing will
 25 initially proceed to step s225 where the query phoneme loop pointer $j2$ is incremented by one.

The processing then returns to step s221 where the incremented value of $j2$ is compared with mx_j . If $j2$ is
 30 less than mx_j , then the processing returns to step s223 and the processing proceeds to step s227, which is operable to prevent too large a hop along both the sequences of annotation phonemes and query phonemes. It does this by ensuring that the path is only propagated if

$i2 + j2$ is less than $i + j + mxhops$. This ensures that only the triangular set of points shown in Figure 11 are processed. Provided this condition is met, the processing then proceeds to step s229 where the system calculates the transition score (TRANSCORE) from lattice point (i,j) to lattice point $(i2,j2)$. The processing then proceeds to step s231 where the system adds the transition score determined in step s229 to the cumulative score stored for the point (i,j) and copies this to a temporary store, TEMPSCORE. As mentioned above, in this embodiment, if two or more dynamic programming paths meet at the same lattice point, the cumulative scores associated with each of the paths are added together. Therefore, in step s233, the system performs a log addition of TEMPSCORE with the cumulative score already stored for point $(i2,j2)$ and the result is stored in SCORE $(i2,j2)$. The processing then returns to step s225 where the query phoneme loop pointer $j2$ is incremented by one and the processing returns to step s221. Once the query phoneme loop pointer $j2$ has reached the value of mxj , the processing proceeds to step s235, where the query phoneme loop pointer $j2$ is reset to the initial value j and the annotation phoneme loop pointer $i2$ is incremented by one. The processing then proceeds to step s219 where the processing begins again for the next column of points shown in Figure 11. Once the path has been propagated from point (i,j) to all the other points shown in Figure 11, the processing ends.

If the decision blocks s213 and s215 determine that the annotation is text and the query is speech, then the processing proceeds to steps s241 to s251, which are operable to propagate the path ending at point (i,j) to the points shown in Figure 9a. In particular, in step

s241, the system determines whether or not the annotation phoneme loop pointer i is pointing to the last phoneme in the annotation. If it is, then there are no more phonemes in the annotation and the processing ends. If
 5 the annotation phoneme loop pointer i is less than $Nann-1$, then the processing proceeds to step s243, where the query phoneme loop pointer $j2$ is compared with mxj . Initially, $j2$ will be less than mxj and therefore the processing proceeds to step s245 where the system
 10 calculates the transition score (TRANSCORE) from point (i,j) to point $(i+1,j2)$. This transition score is then added to the cumulative score associated with the path ending at point (i,j) and the result is copied to the temporary score, TEMPSCORE. The system then performs, in
 15 step s249, a log addition of TEMPSCORE with the cumulative score associated with the point $(i+1,j2)$ and stores the result in SCORE $(i+1,j2)$, to ensure that the path scores for paths which meet at the lattice point $(i+1,j2)$ are combined. The processing then proceeds to
 20 step s251 where the query phoneme loop pointer $j2$ is incremented by one and then the processing returns to step s243. Once the path which ends at point (i,j) has been propagated to the other points shown in Figure 9a, $j2$ will equal mxj and the propagation of the path ending
 25 at point (i,j) will end.

If the decision blocks s213 and s217 determine that the annotation is speech and the query is text, then the processing proceeds to steps s255 to s265 shown in Figure
 30 16b, which are operable to propagate the path ending at point (i,j) to the other points shown in Figure 9b. This is achieved by firstly checking, in step s255, that the query phoneme loop pointer j is not pointing to the last phoneme in the sequence of phonemes representing the

query. If it is not, then the processing proceeds to step s257 where the annotation phoneme loop pointer i2 is compared with mxi. Initially i2 has a value of i and provided annotation phoneme i is not at the end of the sequence of phonemes representing the annotation, the processing will proceed to step s259, where the transition score for moving from point (i,j) to point (i2,j+1) is calculated. The processing then proceeds to step s261 where this transition score is added to the cumulative score for the path ending at point (i,j) and the result is copied to the temporary score, TEMPSCORE. The processing then proceeds to step s263 where a log addition is performed of TEMPSCORE with the cumulative score already stored for the point (i2,j+1) and the result is stored in SCORE (i2,j+1). The processing then proceeds to step s265 where the annotation phoneme loop pointer i2 is incremented by one and the processing returns to step s257. These processing steps are then repeated until the path ending at point (i,j) has been propagated to each of the other points shown in Figure 9b. At this time, the propagation of the path at point (i,j) is completed and the processing ends.

Finally, if the decision blocks s213 and s215 determine that both the annotation and the query are text, then the processing proceeds to steps s271 to s279 shown in Figure 16b, which are operable to propagate the path ending at point (i,j) to the point (i+1,j+1), provided of course there is a further annotation phoneme and a further query phoneme. In particular, in step s271, the system checks that the annotation phoneme loop pointer i is not pointing to the last annotation phoneme. If it is not then the processing proceeds to step s273 where a similar check is made for the query phoneme loop pointer j

relative to the sequence of query phonemes. If there are no more annotation phonemes or if there are no more query phonemes, then the processing ends. If, however, there is a further annotation phoneme and a further query phoneme, then the processing proceeds to step s275 where the system calculates the transition score from point (i,j) to point (i+1,j+1). This transition score is then added, in step s277, to the cumulative score stored for the point (i,j) and stored in the temporary score, TEMPSCORE. The processing then proceeds to step s279 where the system performs a log addition of TEMPSCORE with the cumulative score already stored for point (i+1,j+1) and the result is copied to SCORE (i+1,j+1). As those skilled in the art will appreciate, steps s277 and s279 are necessary in this embodiment because the dynamic programming constraints allow a path to start at any phoneme within the sequence of phonemes representative of the annotation and therefore, point (i+1,j+1) may already have a score associated with it. After step s279, the propagation of point (i,j) is completed and the processing ends.

Transition Score

In steps s103, s105 and s107 shown in Figure 12, dynamic programming paths are propagated and during this propagation, the transition score from one point to another point is calculated in steps s127, s117, s177, s191, s229, s245, s259 and s275. In these steps, the system calculates the appropriate insertion probabilities, deletion probabilities and decoding probabilities relative to the start point and end point of the transition. The way in which this is achieved in this embodiment, will now be described with reference to Figures 17 and 18.

In particular, Figure 17 shows a flow diagram which illustrates the general processing steps involved in calculating the transition score for a path propagating from lattice point (i,j) to lattice point (i2,j2). In
5 step s291, the system calculates, for each annotation phoneme which is inserted between point (i,j) and point (i2,j2), the score for inserting the inserted phoneme(s) (which is just the log of probability $PI()$ discussed above) and adds this to an appropriate store, INSERTSCORE. The processing then proceeds to step s293
10 where the system performs a similar calculation for each query phoneme which is inserted between point (i,j) and point (i2,j2) and adds this to INSERTSCORE. Note, however, that if (i,j) is the start null node \emptyset_s , or if
15 (i2,j2) is the end null node \emptyset_e , then the system does not calculate the insertion probabilities for any inserted annotation phonemes (since there is no penalty for starting or ending a path at any of the annotation phonemes), although it does calculate insertion
20 probabilities for any inserted query phonemes. As mentioned above, the scores which are calculated are log based probabilities, therefore the addition of the scores in INSERTSCORE corresponds to the multiplication of the corresponding insertion probabilities. The processing
25 then proceeds to step s295 where the system calculates the scores for any deletions and/or any decodings in propagating from point (i,j) to point (i2,j2) and these scores are added and stored in an appropriate store, DELSCORE. The processing then proceeds to step s297
30 where the system adds INSERTSCORE and DELSCORE and copies the result to TRANSORE.

The processing involved in step s295 to determine the deletion and/or decoding scores in propagating from point

(i,j) to point (i2,j2) will now be described in more detail with reference to Figure 18. Since the possible deletions and decodings depend on whether or not the annotation was generated from text and whether or not the query was generated from text, the decision blocks s301, s303 and s305 determine if the annotation is text or speech and if the query is text or speech. If these decision blocks determine that both the annotation and the query are text, then there are no deletions and the decoding of the two phonemes is performed by a boolean match in step s307. If annotation phoneme a_{i2} is the same as query phoneme q_{j2} , then the processing proceeds to step s309, where TRANSCORE is set to equal $\log [one]$ (i.e. zero) and the processing ends. If, however, annotation phoneme a_{i2} is not the same as query phoneme q_{j2} , then the processing proceeds to step s311 where TRANSCORE is set to a very large negative number which is a system approximation of $\log [zero]$ and the processing then ends.

If the decision blocks s301 and s305 determine that the annotation is speech and the query is text, then the transition scores are determined using the simplified form of equation (4) discussed above. In this case, the processing passes from step s303 to step s313 where the system determines if annotation loop pointer i2 equals annotation loop pointer i. If it does, then this means that the path has propagated from point (i,j) to point (i,j+1). Therefore, the query phoneme q_{j+1} has been deleted from the sequence of annotation phonemes relative to the sequence of query phonemes. Therefore, in step s317, the system copies the log probability of deleting phoneme q_{j+1} (i.e. $\log P(\emptyset | q_{j+1}, C)$) to DELSCORE and the processing ends. If in step s313, the system determines

that i_2 is not equal to i , then the system is considering the propagation of the path ending at point (i, j) to one of the points $(i+1, j+1)$, $(i+2, j+1)$ or $(i+3, j+1)$. In which case, there are no deletions, only insertions and a decoding between annotation phoneme a_{i_2} with query phoneme q_{j+1} . Therefore, in step s315, the system copies the log probability of decoding query phoneme q_{j+1} as annotation phoneme a_{i_2} (i.e. $\log P(a_{i_2} | q_{j+1}, C)$) to DELSCORE and the processing ends.

If the decision blocks s301 and s305 determine that the annotation is text and that the query is speech, then the transition scores are determined using the other simplified form of equation (4) discussed above. In this case, the processing passes from step s305 to step s319 where the system determines whether or not query phoneme loop pointer j_2 equals query phoneme loop pointer j . If it does, then the system is calculating the transition score from point (i, j) to point $(i+1, j)$. In this case, the annotation phoneme a_{i+1} has been deleted from the sequence of query phonemes relative to the sequence of annotation phonemes. Therefore, in step s321, the system determines and copies the log probability of deleting annotation phoneme a_{i+1} (i.e. $\log P(\emptyset | a_{i+1}, C)$) to DELSCORE and then the processing ends. If at step s319, the system determines that query phoneme loop pointer j_2 is not equal to query phoneme loop pointer j , then the system is currently determining the transition score from point (i, j) to one of the points $(i+1, j+1)$, $(i+1, j+2)$ or $(i+1, j+3)$. In this case, there are no deletions, only insertions and a decoding between annotation phoneme a_{i+1} with query phoneme q_{j_2} . Therefore, in step s323, the system determines and copies the log probability of decoding annotation phoneme a_{i+1} as query phoneme q_{j_2} (i.e.

$\log P(q_{j2}|a_{i+1},C))$ to DELSCORE and the processing ends.

If the decision blocks s301 and s303 determine that both the annotation and the query are generated from speech,
 5 then the transition scores are determined using equation (4) above. In this case, the processing passes from step s303 to step s325 where the system determines if the annotation loop pointer i2 equals annotation loop pointer i. If it does, then the processing proceeds to step s327
 10 where a phoneme loop pointer r is initialised to one. The phoneme pointer r is used to loop through each possible phoneme known to the system during the calculation of equation (4) above. The processing then proceeds to step s329, where the system compares the
 15 phoneme pointer r with the number of phonemes known to the system, Nphonemes (which in this embodiment equals 43). Initially r is set to one in step s327, therefore the processing proceeds to step s331 where the system determines the log probability of phoneme p_r occurring
 20 (i.e. $\log P(p_r|C)$) and copies this to a temporary score TEMPDELSCORE. If annotation phoneme loop pointer i2 equals annotation phoneme i, then the system is propagating the path ending at point (i,j) to one of the points (i,j+1), (i,j+2) or (i,j+3). Therefore, there is
 25 a phoneme in the query which is not in the annotation. Consequently, in step s333, the system adds the log probability of deleting phoneme p_r from the annotation (i.e. $\log P(\emptyset|p_r,C)$) to TEMPDELSCORE. The processing then proceeds to step s335, where the system adds the log
 30 probability of decoding phoneme p_r as query phoneme q_{j2} (i.e. $\log P(q_{j2}|p_r,C)$) to TEMPDELSCORE. The processing then proceeds to step s337 where the log addition of TEMPDELSCORE and DELSCORE is performed and the result is stored in DELSCORE. The processing then proceeds to step

s339 where the phoneme loop pointer r is incremented by one and then the processing returns to step s329 where a similar processing is performed for the next phoneme known to the system. Once this calculation has been performed for each of the 43 phonemes known to the system, the processing ends.

If at step s325, the system determines that i_2 is not equal to i , then the processing proceeds to step s341 where the system determines if the query phoneme loop pointer j_2 equals query phoneme loop pointer j . If it does, then the processing proceeds to step s343 where the phoneme loop pointer r is initialised to one. The processing then proceeds to step s345 where the phoneme loop pointer r is compared with the total number of phonemes known to the system (N_{phonemes}). Initially r is set to one in step s343, and therefore, the processing proceeds to step s347 where the log probability of phoneme p_r occurring is determined and copied into the temporary store TEMPDELSCORE. The processing then proceeds to step s349 where the system determines the log probability of decoding phoneme p_r as annotation phoneme a_{i_2} and adds this to TEMPDELSCORE. If the query phoneme loop pointer j_2 equals query phoneme loop pointer j , then the system is propagating the path ending at point (i, j) to one of the points $(i+1, j)$, $(i+2, j)$ or $(i+3, j)$. Therefore, there is a phoneme in the annotation which is not in the query. Consequently, in step s351, the system determines the log probability of deleting phoneme p_r from the query and adds this to TEMPDELSCORE. The processing then proceeds to step s353 where the system performs the log addition of TEMPDELSCORE with DELSCORE and stores the result in DELSCORE. The phoneme loop pointer r is then incremented by one in step s355 and the

processing returns to step s345. Once the processing steps s347 to s353 have been performed for all the phonemes known to the system, the processing ends.

5 If at step s341, the system determines that query phoneme loop pointer j2 is not equal to query phoneme loop pointer j, then the processing proceeds to step s357 where the phoneme loop pointer r is initialised to one. The processing then proceeds to step s359 where the

10 system compares the phoneme counter r with the number of phonemes known to the system (Nphonemes). Initially r is set to one in step s357, and therefore, the processing proceeds to step s361 where the system determines the log probability of phoneme p_r occurring and copies this to

15 the temporary score TEMPDELSCORE. If the query phoneme loop pointer j2 is not equal to query phoneme loop pointer j, then the system is propagating the path ending at point (i,j) to one of the points (i+1,j+1), (i+1,j+2) and (i+2,j+1). Therefore, there are no deletions, only

20 insertions and decodings. The processing therefore proceeds to step s363 where the log probability of decoding phoneme p_r as annotation phoneme a_{i2} is added to TEMPDELSCORE. The processing then proceeds to step s365 where the log probability of decoding phoneme p_r as query

25 phoneme q_{j2} is determined and added to TEMPDELSCORE. The system then performs, in step s367, the log addition of TEMPDELSCORE with DELSCORE and stores the result in DELSCORE. The phoneme counter r is then incremented by one in step s369 and the processing returns to step s359.

30 Once processing steps s361 to s367 have been performed for all the phonemes known to the system, the processing ends.

NORMALISATION

The above description of the dynamic programming process has dealt only with the numerator part of equation (3) above. Therefore, after an input query has been matched with a sequence of annotation phonemes in the database, the score for that match (which is stored in ENDScore) must be normalised by the normalising term defined by the denominator of equation (3). As mentioned above, the calculation of the denominator term is performed at the same time as the calculation of the numerator, i.e. in the dynamic programming routine described above. This is because, as can be seen from a comparison of the numerator and the denominator, the terms required for the denominator are all calculated on the numerator. It should, however, be noted that when the annotation or the query is generated from text, no normalisation is performed. In this embodiment, normalisation is performed so that longer annotations are not given more weight than shorter annotations and so that annotations which include common phonemes are not given more weight than annotations which include uncommon phonemes. It does this by normalising the score by a term which depends upon how well the annotation matches the underlying model.

TRAINING

In the above embodiment, the system used 1892 decoding/deletion probabilities and 43 insertion probabilities (referred to above as the confusion statistics) which were used to score the dynamic programming paths in the phoneme matching operation. In this embodiment, these probabilities are determined in advance during a training session and are stored in a memory (not shown). In particular, during this training

session, a speech recognition system is used to provide a phoneme decoding of speech in two ways. In the first way, the speech recognition system is provided with both the speech and the actual words which are spoken. The speech recognition unit can therefore use this information to generate the canonical phoneme sequence of the spoken words to obtain an ideal decoding of the speech. The speech recognition system is then used to decode the same speech, but this time without knowledge of the actual words spoken (referred to hereinafter as the free decoding). The phoneme sequence generated from the free decoding will differ from the canonical phoneme sequence in the following ways:

- i) the free decoding may make mistakes and insert phonemes into the decoding which are not present in the canonical sequence or, alternatively, omit phonemes in the decoding which are present in the canonical sequence;
- ii) one phoneme may be confused with another; and
- iii) even if the speech recognition system decodes the speech perfectly, the free decoding may nonetheless differ from the canonical decoding due to the differences between conversational pronunciation and canonical pronunciation, e.g., in conversational speech the word "and" (whose canonical forms are /ae/ /n/ /d/ and /ax/ /n/ /d/) is frequently reduced to /ax/ /n/ or even /n/.

Therefore, if a large number of utterances are decoded into their canonical forms and their free decoded forms, then a dynamic programming method can be used to align the two. This provides counts of what was decoded, d, when the phoneme should, canonically, have been a p.

From these training results, the above decoding, deletion and insertion probabilities can be approximated in the following way.

5 The probability that phoneme, d, is an insertion is given by:

$$P(d|C) = \frac{I_d}{n_o^d} \quad (9)$$

10 where I_d is the number of times the automatic speech recognition system inserted phoneme d and n_o^d is the total number of decoded phonemes which are inserted relative to the canonical sequence.

The probability of decoding phoneme p as phoneme d is given by:

15

$$P(d|p,C) = \frac{c_{dp}}{n_p} \quad (10)$$

20 where c_{dp} is the number of times the automatic speech recognition system decoded d when it should have been p and n_p is the number of times the automatic speech recognition system decoded anything (including a deletion) when it should have been p.

25 The probability of not decoding anything (i.e. there being a deletion) when the phoneme p should have been decoded is given by:

$$P(\emptyset|p,C) = \frac{O_p}{n_p} \quad (11)$$

where O_p is the number of times the automatic speech recognition system decoded nothing when it should have decoded p and n_p is the same as above.

5

ALTERNATIVE EMBODIMENTS

As those skilled in the art will appreciate, the above technique for matching one sequence of phonemes with another sequence of phonemes can be applied to applications other than data retrieval. Additionally, as those skilled in the art will appreciate, although the system described above has used phonemes in the phoneme and word lattice, other phoneme-like units can be used, such as syllables or katakana (Japanese alphabet).

15

As those skilled in the art will appreciate, the above description of the dynamic programming matching and alignment of the two sequences of phonemes was given by way of example only and various modifications can be made. For example, whilst a raster scanning technique for propagating the paths through the lattice points was employed, other techniques could be employed which progressively propagates the paths through the lattice points. Additionally, as those skilled in the art will appreciate, dynamic programming constraints other than those described above may be used to control the matching process.

20
25
30

In the above embodiment, the annotation was generally longer than the query and the dynamic programming alignment algorithm aligned the query with the entire

annotation. In an alternative embodiment, the alignment algorithm may compare the query with the annotation by stepping the query over the annotation from beginning to end and, at each step, comparing the query with a portion of the annotation of approximately the same size as the query. In such an embodiment, at each step, the query would be aligned with the corresponding portion of the annotation using a similar dynamic programming technique to the one described above. This technique is illustrated in Figure 19a with the resulting plot showing the way in which the dynamic programming score for the alignments between the query and a current annotation varies as the query is stepped over the annotation shown in Figure 19b. The peaks in the plot shown in Figure 19b represent the parts of the annotation which match best with the query. The annotation which is most similar to the query can then be determined by comparing the peak DP score obtained during the comparison of the query with each annotation.

In the above embodiment, pictures were annotated using the phoneme and word lattice annotation data. As those skilled in the art will appreciate, this phoneme and word lattice data can be used to annotate many different types of data files. For example, this kind of annotation data can be used in medical applications for annotating x-rays of patients, 3D videos of, for example, NMR scans, ultrasound scans etc. It can also be used to annotate 1D data, such as audio data or seismic data.

In the above embodiments, a speech recognition system which generates a sequence of phonemes from the input speech signal was used. As those skilled in the art will appreciate, the above system can be used with other types

of speech recognition systems which generate, for example, a sequence of output words or a word lattice which can then be decomposed into a corresponding string of phonemes with alternatives, in order to simulate a recogniser which produces phoneme strings.

In the above embodiment, the insertion, deletion and decoding probabilities were calculated from the confusion statistics for the speech recognition system using a maximum likelihood estimate of the probabilities. As those skilled in the art of statistics will appreciate, other techniques, such as maximum entropy techniques, can be used to estimate these probabilities. Details of a suitable maximum entropy technique can be found at pages 45 to 52 in the book entitled "Maximum Entropy and Bayesian Methods" published by Kluwer Academic publishers and written by John Skilling, the contents of which are incorporated herein by reference.

In the above embodiment, the database 29 and the automatic speech recognition unit 51 were both located within the user terminal 59. As those skilled in the art will appreciate, this is not essential. Figure 20 illustrates an embodiment in which the database 29 and the search engine 53 are located in a remote server 60 and in which the user terminal 59 accesses the database 29 via the network interface units 67 and 69 and a data network 68 (such as the Internet). In this embodiment, the user terminal 59 can only receive voice queries from the microphone 7. These queries are converted into phoneme and word data by the automatic speech recognition unit 51. This data is then passed to the control unit 55 which controls the transmission of the data over the data network 68 to the search engine 53 located within the

remote server 60. The search engine 53 then carries out the search in a similar manner to the way in which the search was performed in the above embodiment. The results of the search are then transmitted back from the search engine 53 to the control unit 55 via the data network 68. The control unit 55 then considers the search results received back from the network and displays appropriate data on the display 57 for viewing by the user 39.

In addition to locating the database 29 and the search engine 53 in the remote server 60, it is also possible to locate the automatic speech recognition unit 51 in the remote server 60. Such an embodiment is shown in Figure 21. As shown, in this embodiment, the input voice query from the user is passed via input line 61 to a speech encoding unit 73 which is operable to encode the speech for efficient transfer through the data network 68. The encoded data is then passed to the control unit 55 which transmits the data over the network 68 to the remote server 60, where it is processed by the automatic speech recognition unit 51. The phoneme and word data generated by the speech recognition unit 51 for the input query is then passed to the search engine 53 for use in searching the database 29. The search results generated by the search engine 53 are then passed, via the network interface 69 and the network 68, back to the user terminal 59. The search results received back from the remote server are then passed via the network interface unit 67 to the control unit 55 which analyses the results and generates and displays appropriate data on the display 57 for viewing by the user 39.

In a similar manner, a user terminal 59 may be provided

which only allows typed inputs from the user and which has the search engine and the database located in the remote server. In such an embodiment, the phonetic transcription unit 75 may be located in the remote server 60 as well.

In the above embodiments, a dynamic programming algorithm was used to align the sequence of query phonemes with the sequences of annotation phonemes. As those skilled in the art will appreciate, any alignment technique could be used. For example, a naive technique could be used which identifies all possible alignments. However, dynamic programming is preferred because of its ease of implementation using standard processing hardware.

A description has been given above of the way in which two or more canonical sequences of phonemes are compared using a dynamic programming technique. However, as shown in Figures 2 and 3, the annotations are preferably stored as lattices. As those skilled in the art will appreciate, in order that the above comparison techniques will work with these lattices, the phoneme sequences defined by the lattices must be "flattened" into a single sequence of phonemes with no branches. A naive approach to do this would be to identify all the different possible phoneme sequences defined by the lattice and then to compare each of those with the or each query sequence. However, this is not preferred, since common parts of the lattice will be matched several times with each query sequence. Therefore, the lattice is preferably flattened by sequentially labelling each phoneme within the lattice in accordance with the time stamp information available for each phoneme within the lattice. Then, during the dynamic programming alignment,

different dynamic programming constraints are used at each DP lattice point, in order to ensure that the paths propagate in accordance with the lattice structure.

5 The table below illustrates the DP constraints used for part of the phoneme lattice shown in Figure 2. In particular, the first column illustrates the phoneme number (p_1 to p_9) assigned to each phoneme in the lattice; the middle column corresponds to the actual phoneme in the lattice; and the last column illustrates, for each phoneme, the phonemes to which a path ending at that phoneme may propagate to, at the next dynamic programming time point. Although not shown, the middle column will also include details of the node to which the phoneme is connected and the corresponding phoneme link.

Phoneme Number	Phoneme	Dynamic Programming Constraints
p_1	/p/	$p_1; p_2; p_3; p_4$
p_2	/ih/	$p_2; p_3; p_4; p_5$
p_3	/k/	p_6 $p_3; p_4; p_5; p_7$ p_8
p_4	/ch/	$p_6; p_{10}$ $p_4; p_5; p_7; p_9$ $p_8; p_{11}$
p_5	/ax/	$p_6; p_{10}; p_{12}$ $p_5; p_7; p_9; p_{12}$ $p_8; p_{11}; p_{13}$ p_{14}
p_6	/ax/	$p_6; p_{10}; p_{12}; p_{15}$ p_{16}

P ₇	/ao/	P ₇ ; P ₉ ; P ₁₂ ; P ₁₅ P ₁₆
P ₈	/ah/	P ₈ ; P ₁₁ ; P ₁₃ ; P ₁₈ P ₁₄ ; P ₁₇
P ₉	/f/	P ₉ ; P ₁₂ ; P ₁₅ ; P ₁₈ P ₁₆ ; P ₁₈

5 For example, if a dynamic programming path ends at time
ordered phoneme p₄, then that dynamic programming path
can stay at phoneme p₄ or it can propagate to any of time
ordered phonemes p₅ to p₁₁. As shown in the table, at
some points the possible phonemes to which a path can
10 extend are not consecutively arranged in the time ordered
phoneme sequence. For example, for a dynamic programming
path ending at time ordered phoneme p₆, this path can
either stay at this phoneme or progress to phonemes p₁₀,
p₁₂, p₁₅ or p₁₆. By consecutively numbering the phonemes
15 in the lattice in this way and by varying the dynamic
programming constraints used in dependence upon the
lattice, an efficient dynamic programming matching
between the input query and the annotation lattice can be
achieved. Further, as those skilled in the art will
20 appreciate, if the input query also generates a lattice,
then this may be flattened in a similar way and the
dynamic programming constraints adjusted accordingly.

In the above embodiment, the same phoneme confusion
25 probabilities were used for both the annotations and the
queries. As those skilled in the art will appreciate, if
different recognition systems are used to generate these,
then different phoneme confusion probabilities should be
used for the annotations and the queries. Since these
30 confusion probabilities depend upon the recognition

system that was used to generate the phoneme sequences.

In the above embodiment, when either the annotation or the query was generated from text, it was assumed that the canonical sequence of phonemes corresponding to the typed text was correct. This may not be the case since this assumes that the typed word or words are not misspelled or mis-typed. Therefore, in an alternative embodiment, confusion probabilities may also be used for typed queries and/or annotations. In other words, equation (4) would be used even where either the annotation or the query or both are text. The confusion probabilities used may try to codify either or both misspellings and mis-typings. As those skilled in the art will appreciate, the confusion probabilities for mis-typings will depend upon the type of keyboard used. In particular, the confusion probabilities of mis-typing a word will depend upon the layout of the keyboard. For example, if a letter "d" is typed then the keys surrounding the key for the letter "d" will have high mis-typing probabilities whereas those located further away from the "d" key will have lower mis-typing probabilities. As mentioned above, these mis-typing probabilities may be used together with or replaced by confusion probabilities for the mis-spelling of the words. These mis-spelling probabilities may be determined by analysing typed documents from a large number of different users and monitoring the type of misspellings which usually occur. Such mis-spelling probabilities may also take into account transcription errors caused by mis-keying. In such an embodiment, the dynamic programming constraints used should allow for insertions and/or deletions in the typed input. For example, the constraints illustrated in Figure 11 could

be used.

In the first embodiment described above, during the dynamic programming algorithm, equation (4) was
5 calculated for each aligned pair of phonemes. In the calculation of equation (4), the annotation phoneme and the query phoneme were compared with each of the phonemes known to the system. As those skilled in the art will appreciate, for a given annotation phoneme and query
10 phoneme pair, many of the probabilities given in equation (4) will be equal to or very close to zero. Therefore, in an alternative embodiment the annotation and query phoneme pair may only be compared with a subset of all the known phonemes, which subset is determined in advance
15 from the confusion statistics. To implement such an embodiment, the annotation phoneme and the query phoneme could be used to address a lookup table which would identify the model phonemes which need to be compared with the annotation and query phonemes using equation
20 (4).

M-gram

In the systems described above, a dynamic programming algorithm was used to align an entire query against
25 either entire annotations or portions of annotations. In this embodiment, the query is divided into a number of portions and each of the portions is scanned across the annotation to identify corresponding portions in the annotation. This general technique was proposed by Ng,
30 K. and Zue, V.W. and is discussed in, for example, the paper entitled "Subword unit representations for spoken document retrieval" published in the Proceedings of Eurospeech 1997, the contents of which are incorporated herein by reference.

The portion of the query which is scanned over the annotation can be any size. However, if the portion is a single phoneme then there will not be enough discrimination between the annotations, since each phoneme is likely to occur in each annotation a predetermined number of times. Syllable-sized portions, however, are likely to provide more discriminability, although they are not easy to identify in the query. The technique proposed by Ng and Zue is a compromise between these two possibilities. In particular, this technique uses overlapping fixed-size fragments, or M-grams, of the query phoneme sequence to provide a set of features which are then compared with the annotation. This is illustrated in Figure 22 which shows part of a query phoneme string having phonemes $q_1, q_{i+1}, q_{i+2}, q_{i+3}, q_{i+4}$ and q_{i+5} which are split into four M-grams $M_1 (q_1, q_{i+1}, q_{i+2})$, $M_2 (q_{i+1}, q_{i+2}, q_{i+3})$, $M_3 (q_{i+2}, q_{i+3}, q_{i+4})$ and $M_4 (q_{i+3}, q_{i+4}, q_{i+5})$. In this illustration, it is assumed that each of the four M-grams M_1 to M_4 are unique and therefore comprise a unique sequence of three phonemes.

Therefore, in this embodiment, the first step in performing the phoneme search in steps s13 and s21 shown in Figure 5, is to identify all the different M-grams which are in the query phoneme sequence and their frequency of occurrence within the query using, for example, a Boolean search. Then, the search engine 53 determines the frequency of occurrence of the identified M-grams in each of the stored annotations. In this embodiment, to do this, each of the M-grams is aligned along the entire annotation by, effectively, stepping the M-gram along the annotation and, at each step, comparing the current M-gram with a portion of the annotation having the same size as the M-gram. For the M-grams

illustrated in Figure 22, this involves comparing each M-gram with annotation phonemes a_0 , a_1 and a_2 , then with annotation phonemes a_1 , a_2 and a_3 , then with annotation phonemes a_2 , a_3 and a_4 etc. In this embodiment, rather than perform a Boolean comparison at each step, a similar dynamic programming comparison technique to the one described above is used to compare the current M-gram with the current portion of the annotation. In particular, the comparison technique used in this embodiment uses equation (4) given above and the confusion statistics to determine a score for the possible confusion between one of the M-gram phonemes and one of the annotation phonemes during the DP comparison. However, as those skilled in the art will appreciate, the dynamic programming constraints which are used in this embodiment can be simplified because each alignment involves aligning three phonemes of the query with three phonemes of the current annotation.

Once a dynamic programming score has been calculated for each alignment between the current M-gram and the current portion of the current annotation, this score is normalised in a similar manner to the normalisation described above. However, in this embodiment, the score is normalised both with respect to the current portion of the current annotation and with respect to the current M-gram. Additionally, in this embodiment, the normalisation term which is calculated does not take into consideration deletions and insertions relative to the unknown model sequence of phonemes. In other words, for an alignment between annotation phoneme a_i and query phoneme q_j , the system calculates:

$$\frac{\sum_{r=1}^{N_p} P(a_i|p_r C)P(q_j|p_r C)P(p_r|C)}{P(a_i|C)P(q_j|C)} \quad (12)$$

5 This simplification of the normalising term can be done
 in this embodiment because there are only three phonemes
 being matched in the query with three phonemes in the
 annotation and therefore, for each M-gram comparison, it
 is unlikely that there will be any inserted or deleted
 phonemes. However, if the size of the M-grams is
 10 increased then a normalisation term which considers
 insertions and deletions would preferably be used.

Once the current M-gram has been scanned along the
 current annotation, a large number of DP scores will have
 15 been calculated. As those skilled in the art will
 appreciate, the number of DP scores calculated will
 depend upon the length of the current annotation relative
 to the size of the M-gram and the step size (which is
 usually one phoneme). The number of times that the
 20 current M-gram occurs in the current annotation can then
 be found by counting the number of times the DP score
 exceeds a predetermined threshold. This is illustrated
 in Figure 23 which shows the way in which the DP score
 changes as the current M-gram (M_i) is stepped over the
 25 current annotation and the threshold value used. As
 shown, it was found that a threshold value of $2M$, where
 M is the number of phonemes in the M-gram was a suitable
 threshold value. For the example shown in Figure 23, the
 number of occurrences of the current M-gram (M_i) in the
 30 current annotation is five, since the DP score exceeds
 the threshold five times. Once all the different M-grams
 in the input query have been aligned with the current
 annotation, the number of times each M-gram occurs in
 both the current annotation and the query will be known.

This is illustrated in the following table for the four M-grams shown in Figure 22.

5

M-gram	Frequency of Occurrence in Query	Frequency of Occurrence in Current Annotation
M_1	1	0
M_2	2	2
M_3	3	2
M_4	1	1

10

With this information, it is then possible to calculate a similarity score representing the similarity between the input query and the current annotation by comparing the frequencies of occurrences. In this embodiment, a multinomial similarity score is determined using the frequencies of occurrence of the identified M-grams in the input query and in the current annotation. The basic premise behind this technique is that the M-grams form a closed set of features and the database of annotations and the query of unknown class provide a classification based scenario in which to work. In this embodiment there are 43 different phonemes and each M-gram has three phonemes. This results in a set size of nearly 80,000 (43^3). However, as will be shown below, in practice it is only necessary to consider the M-grams which are in the current query.

25

Considering the frequencies of occurrence of the M-grams in the query and in the current annotation to be vectors (i.e. considering the second and third columns in the

30

above table as vectors), then this technique assumes that these two vectors were generated by the same multinomial model (θ). Additionally, by considering that the frequency of occurrence of each of the M-grams is independent from the other frequencies, then the probability of generating one of the vectors given the model (θ) is:

$$P(\bar{x}|\theta) = \theta_1^{x_1} \cdot \theta_2^{x_2} \cdot \theta_3^{x_3} \cdot \theta_4^{x_4} \dots \theta_m^{x_m} \quad (13)$$

where \bar{x} is the vector of frequencies of occurrence, x_i is the frequency of occurrence of the i^{th} M-gram, $\theta_1, \theta_2 \dots \theta_m$ are the model parameters and m is the number of unique M-grams in the system (i.e. 43^3). Labelling the vector of frequencies of occurrence of the M-grams in the query as vector \bar{Q} and the vector of the frequencies of occurrence of the M-grams in the current annotation as vector \bar{A} , then from traditional classification theory, a score for the similarity between the two vectors is given from:

$$P(\bar{Q}|\bar{A}) = \frac{\int d\theta P(\bar{Q}|\theta) P(\bar{A}|\theta) P(\theta)}{\int d\theta P(\bar{A}|\theta) P(\theta)} \quad (14)$$

Assuming that the prior probability of the model, $P(\theta)$, is a Dirichlet distribution with parameter α , then this expands to:

$$P(\bar{Q}|\bar{A}) = \frac{\int d\theta \theta_1^{Q_1+A_1+\alpha-1} \cdot \theta_2^{Q_2+A_2+\alpha-1} \dots \theta_m^{Q_m+A_m+\alpha-1}}{\int d\theta \theta_1^{A_1+\alpha-1} \cdot \theta_2^{A_2+\alpha-1} \dots \theta_m^{A_m+\alpha-1}} \quad (15)$$

This can be solved as a series of Beta integrals and gamma functions to give:

$$P(\bar{Q}|\bar{A}) = \prod_{i:Q_i \neq 0} \prod_{j=0}^{Q_i-1} \left[\frac{A_i + j + \alpha}{D + j_s + m\alpha} \right] \quad (16)$$

where A_i is the frequency of occurrence of the i^{th} M-gram in the annotation, j is a loop counter used to ensure that the term in brackets is calculated for each of the occurrences of the i^{th} M-gram in the query, D is the total length of the annotation measured in features (i.e. $D = \sum_{i=1}^M A_i$) and j_s is an index which is incremented at each calculation of the term in square brackets. In this embodiment, m is 43³ and α is taken to be 0.1, following the argument given in the paper by Philip Garner entitled "On Topic Spotting and Dialogue Move Recognition", Computer Speech and Language, '11:275-306, 1997, the contents of which are incorporated herein by reference. Typically α will be between zero and one, but in some circumstances where there is a large amount of prior knowledge, α may be greater than one. As those skilled in the art will appreciate, the term in brackets in equation (16) above is a measure of the probability of the i^{th} M-gram occurring in the annotation. The α and $m\alpha$ terms in this equation are provided in order to ensure that the probability term does not go below a predetermined level when D is small. Therefore, in this embodiment, the similarity measure which is calculated for the similarity between the input query and each annotation is simply the product of the probabilities of the M-grams occurring in the annotation.

To illustrate the calculations performed, the above equation will be evaluated for the frequencies of occurrence given in the table above. Therefore, in this illustration \bar{Q} is the vector [1,1,2,1] and \bar{A} is the

vector $[0,2,2,1]$. Therefore, the similarity score $(P(\bar{Q}|\bar{A}))$ for the current query and current annotation is:

$$\begin{aligned}
 P(\bar{Q}|\bar{A}) &= \frac{0+0+0.1}{5+0+43^3 \times 0.1} \times \\
 &\quad \frac{1+0+0.1}{5+1+43^3 \times 0.1} \times \\
 &\quad \frac{3+0+0.1}{5+2+43^3 \times 0.1} \times \quad (17) \\
 &\quad \frac{3+1+0.1}{5+3+43^3 \times 0.1} \times \\
 &\quad \frac{1+0+0.1}{5+4+43^3 \times 0.1}
 \end{aligned}$$

This score is then associated with the current annotation and stored until the end of the search. Once the input query has been compared with all the annotations in a similar manner, the control unit considers the scores and identifies the annotation which most likely matches the query as the one having the largest score.

In the above embodiment, M-grams were identified in the query and the frequency of occurrence of these M-grams were calculated by sliding each of the M-grams over the annotation and by performing a dynamic programming alignment operation at each step. Other less computationally intensive techniques could be used to determine the frequencies of occurrence of the M-grams in the query and the annotation. For example, a Boolean match operation could be performed which finds exact matches between the M-grams and corresponding portions of

phonemes in the M-gram with the corresponding portion of the annotation on a one-to-one basis and the probability of confusing the aligned annotation phoneme with the corresponding M-gram phoneme may then be calculated using the confusion statistics, without summing over all known phonemes (i.e. without using equation 4 above).

In the above embodiments, equal-sized M-grams were extracted from the sequence of query phonemes. As those skilled in the art will appreciate different sizes of M-grams may be extracted from the same query. However, such an embodiment is not preferred, because of the added complexity involved.

In the above embodiment, each M-gram was stepped or scanned across each annotation from beginning to end. As those skilled in the art will appreciate, it is not important how or in what order each M-gram is compared with the portions of the annotation. What is important is to identify, in a consistent manner, the number of times each M-gram occurs in the query and in each of the annotations.

In the above embodiment, the system identified a number of different M-grams in the input query and then determined the frequency of occurrence of those M-grams in the annotation. As those skilled in the art will appreciate, when each of the annotations is loaded into the database, the system may scan the annotations to identify the frequency of occurrence of each possible M-gram and store this vector of numbers which can then be used in a subsequent retrieval operations.

In the above embodiment, the similarity between the query and each annotation was determined by taking a product of probability measures as defined in equation (16). As those skilled in the art will appreciate, probability measures other than the one defined in square brackets in equation (16) may be used. For example, the system may calculate A_1/D as the probability measure and adjust this using some predetermined constant if this probability is unreliable because the current annotation is short and therefore does not comprise many M-grams.

In the above embodiments, the features of the annotation and the query which have been aligned and matched have represented units of speech. As those skilled in the art will appreciate, the above-described technique can be used in other applications where the features of the query and the annotation may be confusable due to the inaccuracies of a recognition system which generated the sequences of features. For example, the above technique could be used in optical character or handwriting recognition systems where there is a likelihood that a recognition system might mistake one input character for another.

A number of embodiments and modifications have been described above. As those skilled in the art will appreciate, there are many other embodiments and modifications which will be apparent to those skilled in the art.

Claims:

1. A feature comparison apparatus comprising:

5 means for receiving first and second sequences of features;

means for identifying a plurality of different first sub-sequences of features within said first sequence of features;

10 first obtaining means for obtaining the number of times each of said different first sub-sequences occurs within said first sequence of features:

second obtaining means for obtaining the number of times each of said different first sub-sequences occurs within said second sequence of features; and

15 means for computing a similarity score representative of the similarity between the first and second sequences of features using the numbers obtained from said first and second obtaining means;

characterised in that

20 the apparatus further comprises third obtaining means for obtaining the total number of sub-sequences of features in said second sequence; and in that

said computing means comprises:

25 first computing means for computing a measure of the probability of each of said first sub-sequences occurring

in said second sequence of features using the numbers obtained from said second obtaining means and the number obtained from said third obtaining means; and

5 second computing means for computing said similarity score by taking products of said computed probability measures in dependence upon said numbers obtained from said first obtaining means.

10 2. An apparatus according to claim 1, wherein the probability measure which is computed for each first sub-sequence occurring in said second sequence by said first computing means is proportional to the number of times the first sub-sequence occurs within said second sequence obtained from said second obtaining means and is
15 inversely proportional to the total number of sub-sequences of features in the second sequence obtained from said third obtaining means.

20 3. An apparatus according to claim 1 or 2, wherein said computing means is operable to compute the similarity measure by calculating:

$$\prod_{i:Q_i \neq 0} \prod_{j=0}^{Q_i-1} \left[\frac{A_i + j + \alpha}{D + j_s + m\alpha} \right]$$

where the term in brackets is the probability measure calculated by said first computing means for the i^{th} sub-sequence; A_i is the number of times the i^{th} sub-sequence occurs in the second sequence of features; j is a loop counter used to ensure that the probability measure in brackets is multiplied for each of the occurrences of the i^{th} sub-sequence in the first sequence of features; D is the total number of sub-sequences of features in the second sequence of features obtained by said third obtaining means; j_s is an index which is incremented at each calculation of the probability measure in brackets; and α and $m\alpha$ are constants to ensure that the probability measure in square brackets does not go below a predetermined lower limit.

4. An apparatus according to claim 3, wherein α lies between zero and one.

5. An apparatus according to any preceding claim, wherein said first obtaining means comprises means for performing a Boolean match between each first sub-sequence of features and the first sequence of features and means for incrementing a count associated with a current first sub-sequence of features each time the current first sub-sequence of features matches a sub-

sequence of features in the first sequence of features.

6. An apparatus according to any preceding claim,
wherein said second obtaining means comprises means for
5 performing a Boolean match between each first sub-
sequence of features and the second sequence of features
and means for incrementing a count associated with a
current first sub-sequence of features each time the
current first sub-sequence of features matches a sub-
10 sequence of features in the second sequence of features.

7. An apparatus according to any preceding claim,
further comprising means for defining a plurality of
second sub-sequences of features from said second
15 sequence of features and wherein said second obtaining
means is operable to obtain said numbers by comparing
each first sub-sequence of features with each second sub-
sequence of features.

20 8. An apparatus according to claim 7, wherein said
second obtaining means comprises;

means for comparing a current first sub-sequence of
features with each second sub-sequence of features using
predetermined data including similarity information which
25 defines similarities between different features to

provide a set of similarity measures;

means for thresholding each similarity measure of the set with a predetermined threshold value and for outputting a threshold result; and

5 means for incrementing a count associated with the current sub-sequence of features in dependence upon said threshold result.

10 9. An apparatus according to claim 8, wherein said second obtaining means further comprises means for aligning features of the current first sub-sequence of features with features of a current second sub-sequence of features to form a number of aligned pairs of features; wherein said comparing means is operable to
15 compare the features of each aligned pair of features using said similarity information to generate a comparison score representative of the similarity between the aligned pair of features; and wherein said comparing means further comprises means for combining the
20 comparison scores for all the aligned pairs of features to provide a measure of the similarity between the current first sub-sequence of features and the current second sub-sequence of features.

25 10. An apparatus according to claim 9, wherein said

comparing means comprises:

5 first comparing means for comparing, for each aligned pair, the first sub-sequence feature in the aligned pair with each of a plurality of features taken from a set of predetermined features to provide a corresponding plurality of intermediate comparison scores representative of the similarity between said first sub-sequence feature and the respective features from the set;

10 second comparing means for comparing, for each aligned pair, the second sub-sequence feature in the aligned pair with each of said plurality of features from the set to provide a further corresponding plurality of intermediate comparison scores representative of the similarity between said second sub-sequence feature and the respective features from the set; and

15 means for calculating said comparison score for the aligned pair by combining said pluralities of intermediate comparison scores.

20

11. An apparatus according to claim 10, wherein said first and second comparing means are operable to compare the first sub-sequence feature and the second sub-sequence feature of the aligned pair respectively with each of the features in said set of predetermined

25

features.

12. An apparatus according to claim 10 or 11, wherein said comparing means is operable to generate a comparison score for an aligned pair of features which represents a probability of confusing the second sub-sequence feature of the aligned pair as the first sub-sequence feature of the aligned pair.

13. An apparatus according to claim 12, wherein said first and second comparing means are operable to provide intermediate comparison scores which are indicative of a probability of confusing the corresponding feature taken from the set of predetermined features as the feature in the aligned pair.

14. An apparatus according to claim 13, wherein said calculating means is operable (i) to multiply the intermediate scores obtained when comparing the first and second sub-sequence features in the aligned pair with the same feature from the set to provide a plurality of multiplied intermediate comparison scores; and (ii) to add the resulting multiplied intermediate scores, to calculate said comparison score for the aligned pair.

15. An apparatus according to claim 14, wherein each of said features in said set of predetermined features has a predetermined probability of occurring within a sequence of features and wherein said calculating means
 5 is operable to weigh each of said multiplied intermediate comparison scores with the respective probability of occurrence for the feature from the set used to generate the multiplied intermediate comparison scores.

10 16. An apparatus according to claim 15, wherein said calculating means is operable to calculate:

$$\sum_{r=1}^n P(q_j|p_r)P(a_i|p_r)P(p_r)$$

where q_j and a_i are the aligned pair of first and second sub-sequence features respectively; $P(q_j|p_r)$ is the
 15 probability of confusing set feature p_r as first sub-sequence feature q_j ; $P(a_i|p_r)$ is the probability of confusing set feature p_r as second sub-sequence feature a_i ; and $P(p_r)$ represents the probability of set feature p_r occurring in a sequence of features.

20

17. An apparatus according to claim 16, wherein the confusion probabilities for the first and second sub-sequence features are determined in advance and depend

upon the recognition system that was used to generate the respective first and second sequences.

18. An apparatus according to any of claims 14 to 17,
5 wherein said intermediate scores represent log probabilities and wherein said calculating means is operable to perform said multiplication by adding the respective intermediate scores and is operable to perform said addition of said multiplied scores by performing a
10 log addition calculation.

19. An apparatus according to claim 18, wherein said combining means is operable to add the comparison scores for all the aligned pairs to determine said similarity
15 measure.

20. An apparatus according to any of claims 9 to 19, wherein said aligning means is operable to identify feature deletions and insertions in said first and second
20 sequences of features and wherein said comparing means is operable to generate said comparison score for an aligned pair of features in dependence upon feature deletions and insertions identified by said aligning means which occur in the vicinity of the features in the aligned pair.

21. An apparatus according to any of claims 9 to 20, wherein said aligning means comprises dynamic programming means for aligning said first and second sequences of features using a dynamic programming technique.

5

22. An apparatus according to claim 21, wherein said dynamic programming means is operable to determine progressively a plurality of possible alignments between said current first sub-sequence of features and said current second sub-sequence of features and wherein said
10 comparing means is operable to determine a comparison score for each of the possible aligned pairs of features determined by said dynamic programming means.

23. An apparatus according to claim 22, wherein said
15 comparing means is operable to generate said comparison score during the progressive determination of said possible alignments.

24. An apparatus according to claim 21, 22 or 23,
20 wherein said dynamic programming means is operable to determine an optimum alignment between said current first sub-sequence of features and said current second sub-sequence of features and wherein said combining means is
25 operable to provide said similarity measure by combining

the comparison scores only for the optimum aligned pairs of features.

25. An apparatus according to claim 22 or 23, wherein
5 said combining means is operable to provide said similarity measure by combining all the comparison scores for all the possible aligned pairs of features.

26. An apparatus according to any of claims 10 to 25,
10 wherein each of the features in said first and second sub-sequences of features belongs to said set of predetermined features and wherein said similarity information comprises, for each feature in the set of features, a probability for confusing that feature with
15 each of the other features in the set of features.

27. An apparatus according to claim 26, wherein said confusion probabilities are determined in advance and depend upon the system or systems used to generate the
20 first and second sub-sequences of features.

28. An apparatus according to claim 26 or 27, wherein said predetermined data further includes, for each feature in the set of features, a probability of
25 inserting the feature in a sequence of features.

29. An apparatus according to claim 26, 27 or 28, wherein said predetermined data further includes, for each feature in the set of features, a probability of deleting the feature from a sequence of features.

5

30. An apparatus according to any of claims 9 to 29, wherein said second computing means further comprises normalising means for normalising each of said similarity measures.

10

31. An apparatus according to claim 30, wherein said normalising means is operable to normalise each similarity measure by dividing each similarity measure by a respective normalisation score which varies in dependence upon the length of the corresponding first and second sub-sequences of features.

15

32. An apparatus according to claim 30 or 31, wherein the respective normalisation scores vary in dependence upon the sequence of features in the corresponding first and second sub-sequences of features.

20

33. An apparatus according to any of claims 30 to 32, wherein said dynamic programming means is operable to determine progressively a plurality of possible

25

alignments between said current first sub-sequence of features and said current second sub-sequence of features and wherein said comparing means is operable to determine a comparison score for each of the possible aligned pairs of features determined by said dynamic programming means and wherein said normalising means is operable to calculate the respective normalisation scores during the progressive calculation of said possible alignments by said dynamic programming means.

10

34. An apparatus according to any of claims 7 to 33, wherein said defining means is operable to define said plurality of second sub-sequences as successive portions of the second sequence of features.

15

35. An apparatus according to claim 34, wherein said successive portions are separated from each other by a single feature.

20

36. An apparatus according to any preceding claim, wherein each of said first sub-sequences comprises the same number of features.

25

37. An apparatus according to claim 7 or any claim dependent thereon, wherein each of said second sub-

sequences comprises the same number of features.

38. An apparatus according to claim 37, wherein said
second sub-sequences of features comprise the same number
5 of features as said first sub-sequences of features.

39. An apparatus according to claim 3 or any claim
dependent thereon, wherein each of the features in said
first and second sequences of features belongs to a set
10 of predetermined features and wherein m is the number of
possible sub-sequences of features which can be formed
from the set of predetermined features.

40. An apparatus according to any preceding claim,
15 wherein said first and second sequences of features
represent time sequential signals.

41. An apparatus according to any preceding claim,
wherein said first and second sequences of features
20 represent audio signals.

42. An apparatus according to claim 41, wherein said
first and second sequences of features represent speech.

25 43. An apparatus according to claim 42, wherein each of

said features represents a sub-word unit of speech.

44. An apparatus according to claim 43, wherein each of said features represent a phoneme.

5

45. An apparatus according to any preceding claim, wherein said first sequence of features comprises a plurality of sub-word units generated from a typed input and wherein said similarity information comprises mis-
10 typing probabilities and/or mis-spelling probabilities.

46. An apparatus according to any preceding claim, wherein said second sequence of features comprises a sequence of sub-word units generated from a spoken input
15 and wherein said similarity information comprises mis-recognition probabilities.

47. An apparatus according to any preceding claim, wherein said receiving means is operable to receive a
20 plurality of second sequences of features, wherein said second determining means is operable to determine and output the number of times each of said first sub-sequences of features occurs within each of said second sequences of features and wherein said computing means is
25 operable to compute a respective measure of the

similarity between the first sequence of features and said plurality of second sequences of features.

48. An apparatus according to claim 47, further comprising means for comparing said plurality of similarity measures output by said computing means and for outputting a signal indicative of the second sequence of features which is most similar to said first sequence of features.

49. An apparatus for searching a database comprising a plurality of information entries to identify information to be retrieved therefrom, each of said plurality of information entries having an associated annotation comprising a sequence of features, the apparatus comprising:

means for receiving an input query comprising a sequence of features;

an apparatus according to any of claims 1 to 48 for comparing the query sequence of features with the features of each annotation to provide a set of comparison scores; and

means for identifying said information to be retrieved from said database using said comparison scores.

50. An apparatus for searching a database comprising a plurality of information entries to identify information to be retrieved therefrom, each of said plurality of information entries having an associated annotation comprising a sequence of speech features, the apparatus
5 comprising:

means for receiving an input query comprising a sequence of speech features;

an apparatus according to any of claims 1 to 48 for
10 comparing said query sequence of speech features with the speech features of each annotation to provide a set of comparison results; and

means for identifying said information to be
retrieved from said database using said comparison
15 results;

characterised in that said apparatus according to any of claims 1 to 48 has a plurality of different comparison modes of operation and in that the apparatus further comprises:

20 means for determining (i) if the query sequence of speech features was generated from an audio signal or from text; and (ii) if the sequence of speech features of a current annotation was generated from an audio signal or from text, and for outputting a determination result;
25 and

means for selecting, for the current annotation, the mode of operation of said apparatus according to any of claims 1 to 48 in dependence upon said determination result.

5

51. A feature comparison method comprising the steps of:
receiving first and second sequences of features;
identifying a plurality of different first sub-sequences of features within said first sequence of
10 features;

a first obtaining step of obtaining the number of times each of said different first sub-sequences occurs within said first sequence of features:

a second obtaining step of obtaining the number of
15 times each of said different first sub-sequences occurs within said second sequence of features; and

computing a similarity score representative of the similarity between the first and second sequences of features using the numbers obtained from said first and
20 second obtaining steps;

characterised in that

the method further comprises a third obtaining step of obtaining the total number of sub-sequences of features in said second sequence; and in that

25 said computing step comprises;

a first computing step of computing a measure of the probability of each of said first sub-sequences occurring in said second sequence of features using the numbers obtained from said second obtaining step and the number
 5 obtained from said third obtaining step; and

a second computing step of computing said similarity score by taking products of said computed probability measures in dependence upon said numbers obtained from said first obtaining step.

10

52. A method according to claim 51, wherein the probability measure which is computed for each first sub-sequence occurring in said second sequence in said first computing step is proportional to the number of times the first sub-sequence occurs within said second sequence
 15 obtained in said second obtaining step and is inversely proportional to the total number of sub-sequences of features in the second sequence obtained in said third obtaining step.

20

53. A method according to claim 51 or 52, wherein said computing step computes the similarity measure by calculating:

$$25 \quad \prod_{i:Q_i \neq 0} \prod_{j=0}^{Q_i-1} \left[\frac{A_i + j + \alpha}{D + j_s + m\alpha} \right]$$

where the term in brackets is the probability measure calculated in said first computing step for the i^{th} sub-sequence; A_i is the number of times the i^{th} sub-sequence occurs in the second sequence of features; j is a loop counter used to ensure that the probability measure in brackets is multiplied for each of the occurrences of the i^{th} sub-sequence in the first sequence of features; D is the total number of sub-sequences of features in the second sequence of features obtained in said third obtaining step; j_s is an index which is incremented at each calculation of the probability measure in brackets; and α and $m\alpha$ are constants to ensure that the probability measure in square brackets does not go below a predetermined lower limit.

15

54. A method according to claim 53, wherein α lies between zero and one.

20

55. A method according to any of claims 51 to 54, wherein said first obtaining step comprises the step of performing a Boolean match between each first sub-sequence of features and the first sequence of features and the step of incrementing a count associated with a current first sub-sequence of features each time the current first sub-sequence of features matches a sub-

25

sequence of features in the first sequence of features.

56. A method according to any of claims 51 to 55,
wherein said second obtaining step comprises the step of
5 performing a Boolean match between each first sub-
sequence of features and the second sequence of features
and the step of incrementing a count associated with a
current first sub-sequence of features each time the
current first sub-sequence of features matches a sub-
10 sequence of features in the second sequence of features.

57. A method according to any of claims 51 to 56,
further comprising the step of defining a plurality of
second sub-sequences of features from said second
15 sequence of features and wherein said second obtaining
step obtains said numbers by comparing each first sub-
sequence of features with each second sub-sequence of
features.

20 58. A method according to claim 57, wherein said second
obtaining step comprises the steps of;

comparing a current first sub-sequence of features
with each second sub-sequence of features using
predetermined data including similarity information which
25 defines similarities between different features to

provide a set of similarity measures;

thresholding each similarity measure of the set with a predetermined threshold value and outputting a threshold result; and

5 incrementing a count associated with the current sub-sequence of features in dependence upon said threshold result.

59. A method according to claim 58, wherein said second
10 obtaining step further comprises the step of aligning features of the current first sub-sequence of features with features of a current second sub-sequence of features to form a number of aligned pairs of features; wherein said comparing step compares the features of each
15 aligned pair of features using said similarity information to generate a comparison score representative of the similarity between the aligned pair of features; and wherein said comparing step further comprises the step of combining the comparison scores for all the
20 aligned pairs of features to provide a measure of the similarity between the current first sub-sequence of features and the current second sub-sequence of features.

60. A method according to claim 59, wherein said
25 comparing step comprises:

a first comparing step of comparing, for each aligned pair, the first sub-sequence feature in the aligned pair with each of a plurality of features taken from a set of predetermined features to provide a corresponding plurality of intermediate comparison scores representative of the similarity between said first sub-sequence feature and the respective features from the set;

a second comparing step of comparing, for each aligned pair, the second sub-sequence feature in the aligned pair with each of said plurality of features from the set to provide a further corresponding plurality of intermediate comparison scores representative of the similarity between said second sub-sequence feature and the respective features from the set; and

the step of calculating said comparison score for the aligned pair by combining said pluralities of intermediate comparison scores.

61. A method according to claim 60, wherein said first and second comparing steps compare the first sub-sequence feature and the second sub-sequence feature of the aligned pair respectively with each of the features in said set of predetermined features.

62. A method according to claim 60 or 61, wherein said
comparing step generates a comparison score for an
aligned pair of features which represents a probability
of confusing the second sub-sequence feature of the
5 aligned pair as the first sub-sequence feature of the
aligned pair.

63. A method according to claim 62, wherein said first
and second comparing steps provide intermediate
10 comparison scores which are indicative of a probability
of confusing the corresponding feature taken from the set
of predetermined features as the feature in the aligned
pair.

64. A method according to claim 63, wherein said
calculating step (i) multiplies the intermediate scores
obtained when comparing the first and second sub-sequence
features in the aligned pair with the same feature from
the set to provide a plurality of multiplied intermediate
20 comparison scores; and (ii) adds the resulting multiplied
intermediate scores, to calculate said comparison score
for the aligned pair.

65. A method according to claim 64, wherein each of said
25 features in said set of predetermined features has a

predetermined probability of occurring within a sequence of features and wherein said calculating step weighs each of said multiplied intermediate comparison scores with the respective probability of occurrence for the feature from the set used to generate the multiplied intermediate comparison scores.

66. A method according to claim 65, wherein said calculating step calculates:

$$\sum_{r=1}^n P(q_j|p_r)P(a_i|p_r)P(p_r)$$

where q_j and a_i are the aligned pair of first and second sub-sequence features respectively; $P(q_j|p_r)$ is the probability of confusing set feature p_r as first sub-sequence feature q_j ; $P(a_i|p_r)$ is the probability of confusing set feature p_r as second sub-sequence feature a_i ; and $P(p_r)$ represents the probability of set feature p_r occurring in a sequence of features.

67. A method according to claim 66, wherein the confusion probabilities for the first and second sequence features are determined in advance and depend upon the recognition system that was used to generate the respective first and second sequences.

68. A method according to any of claims 64 to 67, wherein said intermediate scores represent log probabilities and wherein said calculating step performs said multiplication by adding the respective intermediate scores and performs said addition of said multiplied scores by performing a log addition calculation.

69. A method according to claim 68, wherein said combining step adds the comparison scores for all the aligned pairs to determine said similarity measure.

70. A method according to any of claims 59 to 69, wherein said aligning step identifies feature deletions and insertions in said first and second sequences of features and wherein said comparing step generates said comparison score for an aligned pair of features in dependence upon feature deletions and insertions identified by said aligning step which occur in the vicinity of the features in the aligned pair.

71. A method according to any of claims 59 to 70, wherein said aligning step uses a dynamic programming alignment technique for aligning said first and second sequences of features.

72. A method according to claim 71, wherein said dynamic programming step progressively determines a plurality of possible alignments between said current first sub-sequence of features and said current second sub-sequence of features and wherein said comparing step determines a comparison score for each of the possible aligned pairs of features determined by said dynamic programming technique.

73. A method according to claim 72, wherein said comparing step generates said comparison score during the progressive determination of said possible alignments.

74. A method according to claim 71, 72 or 73, wherein said dynamic programming technique determines an optimum alignment between said current first sub-sequence of features and said current second sub-sequence of features and wherein said combining step provides said similarity measure by combining the comparison scores only for the optimum aligned pairs of features.

75. A method according to claim 72 or 73, wherein said combining step provides said similarity measure by combining all the comparison scores for all the possible aligned pairs of features.

76. A method according to any of claims 60 to 75, wherein each of the features in said first and second sub-sequences of features belongs to said set of predetermined features and wherein said similarity
5 information comprises, for each feature in the set of features, a probability for confusing that feature with each of the other features in the set of features.

77. A method according to claim 76, wherein said
10 confusion probabilities are determined in advance and depend upon the system or systems used to generate the first and second sub-sequences of features.

78. A method according to claim 76 or 77, wherein said
15 predetermined data further includes, for each feature in the set of features, a probability of inserting the feature in a sequence of features.

79. A method according to claim 76, 77 or 78, wherein
20 said predetermined data further includes, for each feature in the set of features, a probability of deleting the feature from a sequence of features.

80. A method according to any of claims 59 to 79,
25 wherein said second computing step further comprises a

normalising step for normalising each of said similarity measures.

5 81. A method according to claim 80, wherein said normalising step normalises each similarity measure by dividing each similarity measure by a respective normalisation score which varies in dependence upon the length of the corresponding first and second sub-sequences of features.

10

82. A method according to claim 80 or 81, wherein the respective normalisation scores vary in dependence upon the sequence of features in the corresponding first and second sub-sequences of features.

15

83. A method according to any of claims 80 to 82, wherein said dynamic programming step progressively determines a plurality of possible alignments between said current first sub-sequence of features and said
20 current second sub-sequence of features and wherein said comparing step determines a comparison score for each of the possible aligned pairs of features determined by said dynamic programming technique and wherein said normalising step calculates the respective normalisation
25 scores during the progressive calculation of said

possible alignments by said dynamic programming technique.

5 84. A method according to any of claims 57 to 83, wherein said defining step defines said plurality of second sub-sequences as successive portions of the second sequence of features.

10 85. A method according to claim 84, wherein said successive portions are separated from each other by a single feature.

15 86. A method according to any of claims 51 to 85, wherein each of said first sub-sequences comprises the same number of features.

20 87. A method according to claim 57 or any claim dependent thereon, wherein each of said second sub-sequences comprises the same number of features.

88. A method according to claim 87, wherein said second sub-sequences of features comprise the same number of features as said first sub-sequences of features.

25 89. A method according to claim 53 or any claim

dependent thereon, wherein each of the features in said first and second sequences of features belongs to a set of predetermined features and wherein m is the number of possible sub-sequences of features which can be formed from the set of predetermined features.

5

90. A method according to any of claims 51 to 89, wherein said first and second sequences of features represent time sequential signals.

10

91. A method according to any of claims 51 to 90, wherein said first and second sequences of features represent audio signals.

15

92. A method according to claim 91, wherein said first and second sequences of features represent speech.

93. A method according to claim 92, wherein each of said features represents a sub-word unit of speech.

20

94. A method according to claim 93, wherein each of said features represents a phoneme.

25

95. A method according to claim 58, wherein said first sequence of features comprises a plurality of sub-word

units generated from a typed input and wherein said similarity information comprises mis-typing probabilities and/or mis-spelling probabilities.

5 96. A method according to claim 58, wherein said second sequence of features comprises a sequence of sub-word units generated from a spoken input and wherein said similarity information comprises mis-recognition probabilities.

10

97. A method according to any of claims 51 to 96, wherein said receiving step receives a plurality of second sequences of features, wherein said second determining step determines and outputs the number of
15 times each of said first sub-sequences of features occurs within each of said second sequences of features and wherein said computing step computes a respective measure of the similarity between the first sequence of features and said plurality of second sequences of features.

20

98. A method according to claim 97, further comprising the step of comparing said plurality of similarity measures output by said computing step and outputting a signal indicative of the second sequence of features
25 which is most similar to said first sequence of features.

99. A method of searching a database comprising a plurality of information entries to identify information to be retrieved therefrom, each of said plurality of information entries having an associated annotation comprising a sequence of features, the method comprising the steps of:

receiving an input query comprising a sequence of features;

a method according to any of claims 51 to 98 for comparing the query sequence of features with the features of each annotation to provide a set of comparison scores; and

identifying said information to be retrieved from said database using said comparison scores.

100. A method of searching a database comprising a plurality of information entries to identify information to be retrieved therefrom, each of said plurality of information entries having an associated annotation comprising a sequence of speech features, the method comprising the steps of:

receiving an input query comprising a sequence of speech features;

a method according to any of claims 51 to 98 for

comparing said query sequence of speech features with the speech features of each annotation to provide a set of comparison scores; and

5 means for identifying said information to be retrieved from said database using said comparison results;

characterised in that said method according to any of claims 51 to 98 has a plurality of different comparison modes of operation and in that the method
10 further comprises the steps of:

determining (i) if the query sequence of speech features was generated from an audio signal or from text; and (ii) if the sequence of speech features of a current annotation was generated from an audio signal or from
15 text, and outputting a determination result; and

selecting, for the current annotation, the mode of operation of said method according to any of claims 51 to 98 in dependence upon said determination result.

20 101. A method according to claim 99 or 100, wherein one or more of said information entries is the associated annotation.

102. A method according to any of claims 51 to 101,
25 wherein said method steps are performed in the order in

which they appear in the claim.

103. A storage medium storing processor implementable
instructions for controlling a processor to implement the
5 method of any one of claims 57 to 102.

104. Processor implementable instructions for controlling
a processor to implement the method of any one of claims
51 to 102.



INVESTOR IN PEOPLE

Application No: GB 0025202.3
Claims searched: 1 to 104

106

Examiner: John Donaldson
Date of search: 7 November 2000

Patents Act 1977 Search Report under Section 17

Databases searched:

UK Patent Office collections, including GB, EP, WO & US patent specifications, in:
UK CI (Ed.R): G4R(RHB, RPP, RRL, RRM)
Int CI (Ed.7): G06K 9/00, 9/36, 9/46, 9/54, 9/60, 9/62, 9/64, 9/68, 9/78, 9/80; G10L
15/00, 15/02, 15/08, 15/12, 15/14
Other: Online: WPI, EPODOC, JAPIO

Documents considered to be relevant:

Category	Identity of document and relevant passage	Relevant to claims
A	GB 2302199 A (ALLVOICE), see abstract	-

X	Document indicating lack of novelty or inventive step	A	Document indicating technological background and/or state of the art.
Y	Document indicating lack of inventive step if combined with one or more other documents of same category.	P	Document published on or after the declared priority date but before the filing date of this invention.
&	Member of the same patent family	E	Patent document published on or after, but with priority date earlier than, the filing date of this application.